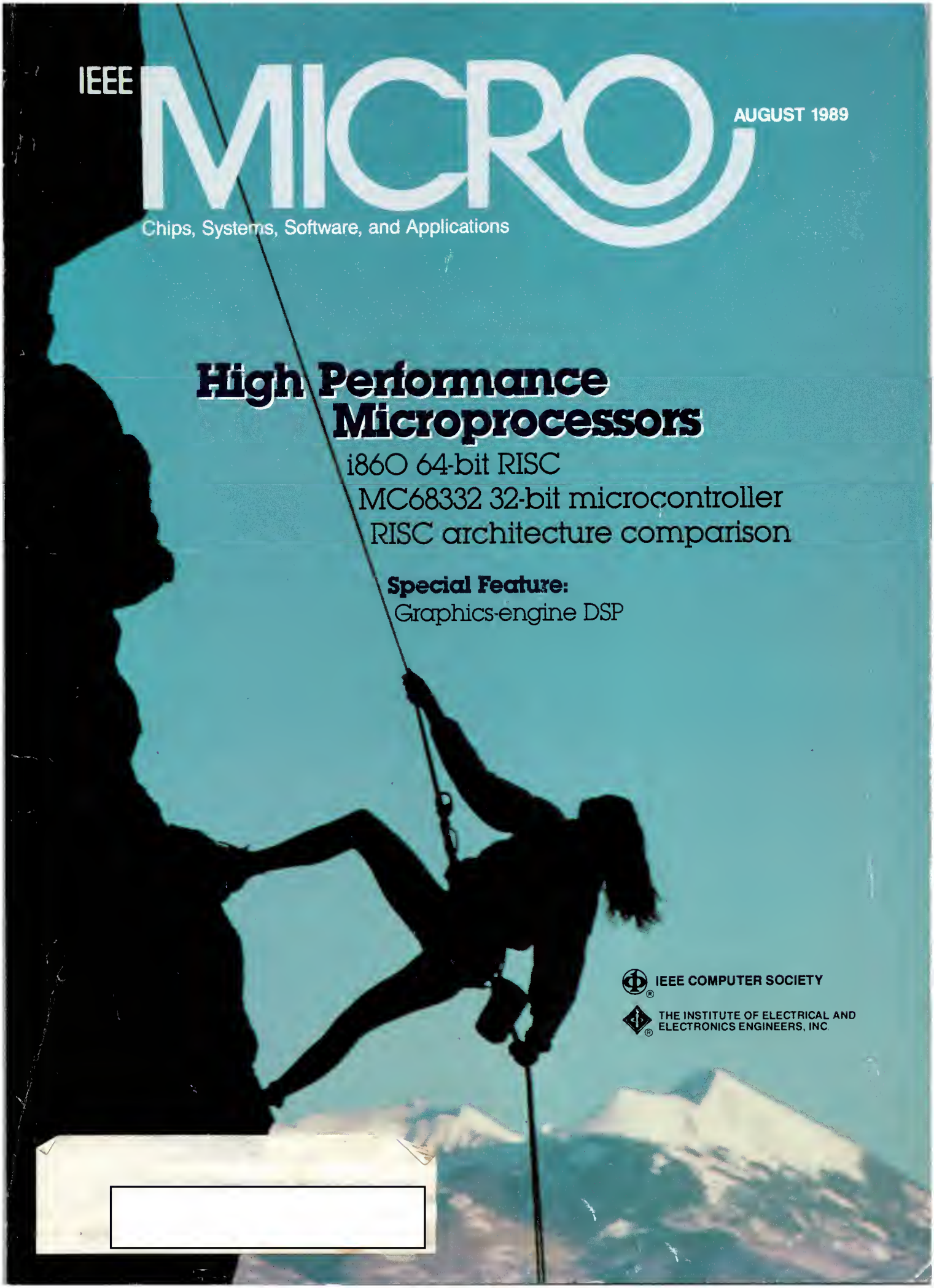
IEEE MICRO

AUGUST 1989

Chips, Systems, Software, and Applications



High Performance Microprocessors

i860 64-bit RISC

MC68332 32-bit microcontroller

RISC architecture comparison

Special Feature:
Graphics-engine DSP

IEEE COMPUTER SOCIETY

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC

Cover image: West Stock, Inc.
Cover design: Design & Direction

IEEE MICRO

Volume 9 Number 4 (ISSN 0272-1732) August 1989

Published by the IEEE Computer Society

Departments

Feature Articles

Special Feature

IEEE Computer Society
10662 Los Vaqueros Circle
Los Alamitos, California 90720
(714) 821–8380

Circulation: *IEEE Micro* (ISSN 0272-1732) is published bimonthly by the IEEE Computer Society, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578; IEEE Computer Society Headquarters, 1730 Massachusetts Ave., NW, Washington, DC 20036-1903; IEEE Headquarters, 345 East 47th St., New York, NY 10017. Annual subscription: $18 in addition to IEEE Computer Society or any other IEEE society member dues; $33 for members of other technical organizations. Back issues: $10 for members; $20 for nonmembers. This journal is also available in microfiche form.

Postmaster: Send address changes and undelivered copies to *IEEE Micro*, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578. Second-class postage is paid at New York, NY, and at additional mailing offices.

IEEE Micro

From the Editor-in-Chief

A glimpse into the future

This issue of *IEEE Micro* is significant in that it covers some of the newest developments in powerful computing chips.

The i860 Intel chip begins the implementation of the concept of a large desktop computer system. The impact of this technology on engineering and science will be substantial, but the impact on medicine, the arts, and business is less well defined.

Having a significant-size computer on every engineer's desk will allow the users to expand their graphical and computing capabilities. Complex problems such as application-specific integrated circuit (ASIC) design with detailed device modeling and solutions to nonlinear problems such as turbulent flow will be solved on a day-to-day basis. Computer-aided design will become the accepted, every-day work method of all engineers.

We can anticipate that in the medical industry such systems as the MRI (magnetic resonance imaging) and CAT (computer-aided tomography) scanners should decrease in cost and increase in functionality. In the area of the arts larger, low-cost machines should offer a new medium for the artist to design and test music, paintings, and sculpture at minimal cost. The demands for very sophisticated, low-cost I/O devices and sophisticated communications interfaces will increase greatly.

The second chip discussed in this issue is the Motorola MC68332. This chip is a new concept in the area of 16/32-bit controllers and embedded processors. Designed so that it can be configured by the user and optimized for a particular task, the Motorola chip decreases application cost and offers more functionality. This processor will find uses in high-performance instrumentation systems, communication systems, graphical systems, and precision-control applications in which cost and functionality are the major factors in the implementation of a controller.

While the newer chips make possible larger, faster, and lower-cost computer workstations, low-cost supporting software is not yet available. The software industry has not really addressed the problem of user software and the licensing of expensive software packages, when the packages will be sold in a high-volume market. The copyrighting of software and software interfaces (to both displays and other software packages) for these new machines and the cost of software development may be the ultimate limiting factors in the use and general adoption of large machines. Micro Law, beginning in the June issue of *Micro* and continuing in this issue, discusses one of the pressing legal problems with the aspects of software development, software design, and user interfaces. The legal issue of who owns and controls software and interfaces will have a more significant impact on the computer industry than all of the hardware developments to date.

Finally, we say good-bye to two editorial board members and hello to three new members. Victor Huang (AT&T Bell Laboratories) has been a board member who has contributed two special issues of *Micro*, one on operating systems and this special issue. Barry Johnson (University of Virginia) has contributed special issues on fault-tolerant computing. Both of these men have reviewed numerous papers and contributed to the overall quality of your magazine, and they will be missed.

Three new board members have been appointed, Jim Tracey, Michael Slater, and Maurice Yunik. Jim Tracey is dean of engineering at the University of Texas at San Antonio and has been involved with computers and logic design for many years. Michael Slater, president of Micro Design Resources Inc.,

In the mailbag

April 1988

"I liked the reviews of advanced MPU chips. I would like to see more reviews of advanced memory chips and memory controllers." K.C., Nepean, Canada

"I liked the design of the 88000 RISC family. I would like to see the addresses of advertisers and product manufacturers." H.U.H., Mankato, MN (We need the information that is on the reader service cards so we do not include the manufacturers' addresses with each new product. This information allows us to do a better job in meeting the requests of our readers for the articles that they want to see in *Micro*.—J.H.)

June 1988

"I liked the issue on embedded processors. I would like to see more of this type of thing but address software issues a bit more." P.M.V., Cedar Rapids, IA (I think you will like the article in this issue on the MC68332 processor. However, there is not much software in the article.—J.H.)

December 1988

"I liked the floating-point DSP (articles). I disliked nothing. I would like to see more about software and algorithms used in computerized tomography." M.G., Lima, Peru

"I liked all articles on floating-point DSP chips and the special feature on PC-based speech spectrograph systems." A.H., Bombay

"I liked the TMS320C30 DSP article and also the other articles. Would like to see practical design circuits based on the above DSPs for experimental purposes." M.S.P., Pune, India

"I liked this issue because of its emphasis on DSP chips." M.S., Mashad, Iran

"I liked articles on DSPs." S.B., Bombay

"I liked the issue. I would like to see a page devoted to applications literature in *Micro*. I would like to receive a reprint copy of 1) The TMS320C30 F-P DSP by Papamichalis and 2) DSP with IEEE-FPA by Sohie." S.O., Trivandrum, India (I have forwarded your request and address to the authors of the articles.—J.H.)

February 1989

"I liked New Products. I would like to see (an article on) the current state of HDTV." C.T., Boca Raton, FL

"I liked the feature articles. I disliked that Micro View started on page 96 and concluded on page 95. Daft!! I would like to see less waffle like New Products and Product Summary. There is enough of it in other journals." S. G., Newcastle upon Tyne, England.

"I would like to see a compendium of AI tools for micros (with) cost and effectiveness." K.L.K., Boulder, CO (A good idea. Do we have a reader who would be willing to make such a survey?—J.H.)

"I liked the robot-arm control, EMMA2 (articles), and the new products. Would like to see issues on transputer applications and programming." T.A.S., Dhahran, Saudi Arabia (There is a lot of interest in the transputer. See the June 1989 *Micro* for an article on the transputer instruction set. We are always interested in receiving papers in the area of transputer development.—J.H.)

"I would like to see more new products." R.T., Aberdeen, Hong Kong

"Please send me your publication free of charge every month." G.F., Mexico (Our major competition is the "free" publications that exist. These "free" publications do cost money, as you can see from the advertising appearing in the publications. *IEEE Micro* does not emphasize advertising. To make the magazine break even in costs, volunteers contribute their time to review submitted manuscripts and make a great many of the editorial decisions. None of the editorial content is paid for except on an emergency basis. I will send you a subscription form for *Micro*.—J.H.)

edits and publishes *Microprocessor Report* and has consented to write our Micro View department beginning with the next issue. Maurice Yunik (University of Manitoba) has worked with human-machine interaction and various aspects of digital signal processing. We are happy to have each new member a part of *IEEE Micro*.

Department

Micro News

Micro News features information of interest to professionals in the microcomputer/microprocessor industry. Send information for inclusion in Micro News one month before cover date to Managing Editor, IEEE Micro, *10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578.*

Intel, DARPA plan 2,000-processor machine

The US Defense Department's Advanced Research Projects Agency has awarded the scientific computer division of Intel Corp. $7.6 million over the next three years for its part in a parallel-processing development called the Touchstone Project. Intel's resulting system is to ultimately contain as many as 2,000 one-million-transistor i860 processors, each with the power of a Cray-1. The target date for prototype development is the end of 1991.

A significant portion of the Touchstone Project grant supports software development to facilitate system use. The total program cost approximates $27.5 million.

A fully configured prototype would contain 128 billion bytes of high-speed memory and more than a terabyte (1,000 gigabytes) of fast file (disk) storage. It would also display animated 3D graphics and connect to high-speed, optical data networks. Sixty-four-bit, floating-point operations are expected to exceed 128 billion/s.

Touchstone software plans incorporate the results of university research in advanced Unix operating systems, reliable disk arrays, shared virtual-memory management, and parallel-programming tools.

Targeted applications include the mathematical solution of sparse matrices and the design of intelligent programs or cooperating expert systems.

Transistors: High- and low-temperature approaches

Researchers at the University of Houston have reported the development of a superconducting equivalent to an electronic transistor. Although years away from becoming a commercial product, the device could possibly allow complex circuits to be designed out of superconducting materials.

The four-terminal device, termed a high-temperature, superconducting, magnetic-field-effect transistor, uses a superconducting wire in a loop running through a material that can be magnetized. Flow of electricity in the loop creates a magnetic field, which regulates other current when it turns on or off.

The inventor of the transistor, Wei-Kan Chu (no relation to C.W. Paul Chu, at the same university), drew inspiration from a superconducting switch called the crossbar cryotron.

At the University of Texas at Austin, A.F. Tasch and S.K. Banerjee have developed a process called remote plasma-enhanced chemical vapor deposition (RPCVD) of silicon. The method reduces by several hundred degrees—to 300° F—the temperature at which thin layers of silicon can be grown. The desired result? Smaller transistors and more of them to a chip.

For a technical report, contact Tasch at (512) 471-1640.

IEEE Standards to launch hypertext series

Because hypertext is especially suited to highly structured technical works, the IEEE Standards Department has chosen this form of electronic publishing for selected standards. (A hypertext document allows easy and intuitive access to dispersed—yet interrelated—information.)

The first project in this series is *IEEE Standard 1003.1-1988, Portable Operating System Interface for Computer Environments* (Posix). Texas Instruments' Hypertrans software was used to prepare this first book. The text and necessary software occupy 5.25-inch, 360-Kbyte disks.

As subsequent Posix standards are published, they will be linked to the first edition to form a collection.

At present, users need an IBM PC or compatible. The Standards Department plans Macintosh or Unix versions if sufficient demand occurs.

Plans indicate release of the book in the fourth quarter of 1989. Readers who would like to furnish input to the project are invited to call Jay Iorio, IEEE Standards Manager of Information Services, at (212) 705-7150.

Screen displays: Federal court follows Computer Society's approach

Richard H. Stern

In September 1987, the US Copyright Office held a public hearing about registration of copyrights for screen displays. At issue was whether to register them as works of authorship independent from related computer programs or as a single work of authorship together with the programs. At that time, judicial decisions conflicted on the issue—although the majority of decisions in the field seemed to favor separate treatment. The IEEE Computer Society's Board of Governors passed a resolution in favor of treating screen displays and computer programs as separate works. A panel representing the society testified at the hearing. [*See* IEEE Micro, *Micro Law, June 1989, p. 84.*]

In June 1988, the Copyright Office published a policy statement rejecting the position of the Computer Society as well as those judicial decisions to the same effect. Henceforth, the Copyright Office would issue only a single registration for computer programs and their associated screen displays. The Society's Committee on Public Policy (COPP) considered whether judicial review should be sought but voted to await further judicial developments before taking any action.

In the first court decision concerning the matter since the Copyright Office announced its new policy, the US District Court for Connecticut held that it would accept the Copyright Office's formal approach. However, it would treat the single copyright registration as if it were two separate registrations—one for the computer program and one for the screen display. It was necessary to do this, the court said, to avoid "difficult, if not insurmountable" problems that the Copyright Office's single-registration policy created in copyright infringement cases.

Manufacturers Technologies, Inc., of Springfield, Massachusetts, sued CAMS, Inc., of Waterbury, Connecticut, for copyright infringement. MTI claimed that CAMS copied the screen displays of MTI's Costimator computer program. This program estimates the cost of machining parts into CAMS' competitive Quick Cost and Rapidcost programs. (MTI also claimed that CAMS copied the code of the Costimator program, but the court rejected that claim and it is not discussed here.)

The court reviewed the prior cases and concluded that a computer program and a screen display are different things, since more than one code can be written to generate the same screen display. However, the court noted that the Copyright Office had decided that the two things could not be registered separately and that only one registration could be issued for both. The court felt that this legal approach, if accepted, would make the plaintiff prove copyright infringement of its computer program. That proof, in turn, would require the plaintiff to show substantial similarity in the two parties' codes as a whole. At least the plaintiff would have to show that the code generating the screen display was such a large portion of the total code in the program that infringement of it (by copying the screen display) amounted to infringing the whole program. By taking that position, the court indicated that using a relatively small fraction of a computer program would not be considered copyright infringement.

The court rejected this approach as unrealistic and unfair to the copyright owner. When a defendant just "reverse-engineered the screen displays themselves" without copying the code, it would probably be impossible to prove any copyright infringement. That approach would effectively deprive creators of screen displays of legal protection. The court said it preferred to create the "legal fiction of two separate registrations," even though only one existed. The court did not want to penalize the plaintiff just because the Copyright Office took an unreasonable approach. The court would rather adopt its own approach, one that "conforms to the realities of the Copyright Office registration procedures."

The court then proceeded to a screen-by-screen analysis of the parties' software packages. It found that some of the 11 registered screens had been infringed and others had not. The main, deciding consideration was whether the allegedly infringed aspects of format were functionally dictated—or arbitrary. All of the plaintiff's claims to "internal methods of navigation" (that is, a choice of keystrokes to move the cursor) were denied:

> . . . to give the plaintiff copyright protection for this aspect of its screen displays would come dangerously close to allowing it to monopolize a significant portion of the easy-to-use internal navigation conventions for computers.

In addition, the court held that CAMS copied the *sequence and flow* of the Costimator screens, which the court considered to be part of the computer program copyright rather than part of the copyright in the individual screens.

The court's legal approach was basically pragmatic. The court agreed with the Computer Society argument: It makes no sense to mix up screen displays with computer programs. But the court recognized that software marketers are stuck with the Copyright Office's administrative rule, and it did not want to penalize them just to straighten out the Copyright Office. The court therefore accepted the Copyright Office's rule, but created a legal fiction under which it would disregard the Copyright Office's rule and "conform to the realities." It is hard to quarrel with that.

Whether the court's individual decisions on particular screens were correct can be determined only by studying each pair of the plaintiff's and the defendant's respective screens. But the court's general principles—trying to protect arbitrary elements of screen-display design and leaving functional elements in the public domain—are certainly sound. The court's treatment of screen sequence and flow is problematical, but the issue is clearly difficult and one on which reasonable minds may differ.

Board appoints three members

Editor-in-Chief Joe Hootman welcomes Michael Slater, James Tracey, and Maurice Yunik to *IEEE Micro*'s Editorial Board.

Slater is editor and publisher of *Microprocessor Report*, a monthly newsletter on microprocessor technology. He also teaches in private industry, consults on design issues, and lectures at Stanford University. Slater was a research and development engineer at Hewlett-Packard. He is the author of the book *Microprocessor-Based Design*, published by Prentice-Hall. Slater received a BSEE degree from the University of California, Berkeley.

Tracey is the dean of the College of Sciences and Engineering at the University of Texas in San Antonio. Previous positions include work at Boeing, IBM, and Rockwell International. Tracey received the BS, MS, and PhD degrees in electrical engineering from Iowa State University. He is a member of the ASEE, ACM, both the Texas and the National Societies of Professional Engineers, and the IEEE. Tracey is a registered professional engineer in the state of Kansas.

Yunik is a professor in the Department of Electrical and Computer Engineering at the University of Manitoba, where he established the computer engineering program. His interests include digital signal processing, computer architecture, and computer music. Yunik received the BSc and MSc degrees in electrical engineering from the University of Manitoa. He is a member of Sigma Xi.

Micro Bits

ARPAnet, a nationwide research network, is being dismantled. Plans call for its eclipse by the end of 1989. The gateway between ARPAnet and NSFnet has been moved to the Pittsburgh Supercomputer Center. Those with connectivity concerns should contact nisc-people@devvax.tn.cornell.edu.

The Technical Committees of the European Committee for Standardization (**CEN**) plans to share working documents with the International Organization for Standardization (**ISO**). The documents are available through ANSI.

Canon and **Next Inc.** plan to distribute the Next Computer in Japan. Canon makes a laser printer and other equipment used with the computer.

The US Defense Department's Advanced Research Projects Agency (**DARPA**) has chosen five companies to share in $30 million for the development of high-definition, TV-display technologies. Newco, Raychem, Texas Instruments, Projectavision, and Photonics Technology are negotiating the exact amounts of the awards.

Ten companies from the computer industry—along with American Airlines—have formed the **Object Management Group, Inc.** OMG promotes industrywide adoption of a common applications object-oriented environment. Companies interested in joining can write OMG at PO Box 395, Westboro, MA 01580.

Current literature

The 88open Consortium Ltd. has published the *Binary Compatibility Standard* for Motorola 88000-based systems. The 215-page BCS specifies interfaces between the binary executable file and the operating system as well as data interchanges for installing software from removable media. *88open Consortium, 8560 SW Salish Lane, Suite 500, Wilsonville, OR 97070.*

Borland International and McGraw-Hill have jointly published two books on the latest version of Turbo Pascal. *Turbo Pascal 5.5: The Complete Reference* is a resource for 5.0 and 5.5 features, commands, and programming techniques at various skill levels. *Turbo Pascal Disktutor* and two disks introduce Version 5.5 and provide a modified Turbo Pascal compiler. *Osborne/McGraw-Hill, 1221 Avenue of the Americas, New York, NY 10020; (212) 512-3851.*

If you're trying to determine where RISCs fit in your life, you might be interested in the *RISC Impact on the Computer Market*. The 204-page report addresses the issues of how existing CISC products may be impacted by RISCs, what software vendors are doing to support specific RISC architectures, and what factors should be considered in selecting a RISC implementation. The analysis also includes the advantages and pitfalls of using a RISC. *Electronic Trend Publications, 12930 Saratoga Ave., Suite D1, Saratoga, CA 95070; (408) 996-7416, ex. 389; $1,250.*

Do you like to laugh it up with shows on bloopers, video outtakes, and vaudevillian pratfalls? Then you might like an audiotaped book called *The World's Greatest Computer Mistakes, Miscues, Flubs, and Foul-ups*. This documentary of bogus nuclear-war alerts, false arrests, killer robots, and ATMs that pour the bank's money onto the street has one central theme. You guessed it: The human error factor in computer mistakes far outweighs the cost of computer crime. *Ned Bulmash, 159 Griggs, Teaneck, NJ 07666; (201) 692-3993.*

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Interest Card.

Low 183 **Medium** 184 **High** 185

Micro Law

Richard H. Stern
Law Offices of Richard H. Stern
1300 19th Street NW, Suite 300
Washington, DC 20036

Appropriate and inappropriate legal protection of user interfaces and screen displays

Part 2, Technical aspects of screen design raising legal policy issues

Do the public importance of innovation and progress in the creation of screen displays and other aspects of computer program user interfaces, and the need to provide incentives to create them, warrant legal protection for their creators? Any grant of such protection raises legitimate concerns that must be addressed before one can justify any conclusion that protection is desirable and in the public interest. These concerns involve the risk of preempting or hindering the work of others in the field.

Good screen design

Important decisions must be made in designing a set of screens for a computer program: How much, and which, information should be placed on a particular screen? It is important not to put so much data on one screen that it becomes cluttered and hard to understand. On the other hand, forcing users to work through too many successive screens may frustrate them or exceed the capabilities of their short-term memory and their attention span.

How should the order in which information and command options is presented to the user be determined? An example shown in the word-processing screen of Figure 1 concerns the selection of one of the entries that will ordinarily cause another screen to appear. Thus, entering the keystroke <P> for the Print command may require the user to make a number of further choices. What document does the user want to print? What portions (pages) of it should be printed; what size page; what spacing between lines; what top, bottom, and left margins? Should the pages be numbered; if so, with what number should the numbering start? Where should the numbers be placed? What font or typeface should be used; how many letters per inch? The screen designer must decide on the order in which to present these choices to the user and how to group them.

Presenting these choices to the user will often require at least one more menu, if not several. There are optimal amounts of information to assign to any single screen, and an optimal number of screen levels. Well-designed screens and screen sequences make it easy for the user to make the necessary choices and tend to prevent mistakes. Poorly designed screens and screen sequences confuse the user, promote mistakes, irritate the user, and make the user dislike the program. Perhaps, poor design will make the user dislike the use of computer programs in general. (This result would impede the proliferation of computers in society and the advancement of various policy goals or agendas of the Computer Society and others interested in promoting computer usage as a means of improving the world.)

Series Highlights

User interfaces of computer programs often determine a product's commercial success. Should the law protect these products against competitive imitation? If so, what is the best mechanism to do so? The US Copyright Office recently ruled that they should be protected but only as an "integrally related" part of their computer programs. Is it not better to protect them separately? What is the best mechanism for doing so? What problems and difficulties might protection cause?

While most of the litigation and legal disputes so far have involved menus, we can reasonably anticipate that future litigation will involve the newer pictorial displays of computer simulations. Owners devote considerable effort to ensuring the user friendliness of screen displays; they want to ensure user acceptance, protect their investment, and secure future profits.

In evaluating the pros and cons of protecting user interfaces and screen displays in this series, I explore the kind of legal rights that might be asserted over the nondevice aspects of these products. In Part 2, I continue with a discussion of the possible difficulties and problems that protection of user interfaces and screen displays might cause. Subsequent issues will present conclusions based on these discussions.

Main Menu

C = Create a new document
E = Edit an existing document
P = Print a document
I = Index of documents on file
D = Delete a document from file
M = More menu selections
Q = Quit using system

Type the right letter and press **Return**.

Figure 1. Screen display of a simple word processing menu.

0272-1732/89/0800-007$01.00 © 1989 IEEE

Screen Design and Human Factors Analysis

Some more specific examples of known techniques of screen design, based on human factors considerations, are the following. At the level of generality used here to describe them, the techniques used in these examples should not be proprietary. (As mentioned in Part 1, I use the convention of representing keystrokes by angle brackets.)

The examples include the practices of:

- *Including at the end of every menu screen the opportunity for the user to go back.* Users should be able to go back to the immediately preceding menu screen, to the main menus, and to quit the set of menus. Users should be able to return to the applications program (for example, a spreadsheet) associated with the menu or to the operating system (for example, the A> prompt on an IBM PC-compatible microcomputer).
- *Using consistent means for carrying out particular functions in the menu.* (Contrast this, for example, with the inconsistency of using <F1> to invoke a help screen in one menu screen and using <F2> for that purpose in another menu screen).
- *Using consistent visual formats for all screens.*
- *Following intuitive command sets.* (For example, using <→> for moving the cursor to the right and <←> for moving the cursor to the left, or using <Escape> to leave a program or a subroutine.)
- *Using conventional keystrokes in command sets in menus* (such as <Page Up> for going back to the preceding screen of the menu or the preceding menu, and <Page Down> for going on to the next screen or menu).
- *Giving the user both command-driven and menu-driven options where possible.* The user might move a cursor to a command line and press <Enter> (a menu-driven operation). Or the user might simply enter a one- or two-letter abbreviation for the command (a command-driven operation).
- *Using customary names for functions.* One example includes Quit for quitting a program, rather than Leave, Depart, Go, or some other nonconventional term. Another is using Save for sending a document from screen and computer memory to diskette, rather than File, Enter, Write, or some other nonconventional term.

The designers of the screens for Crosstalk, the program involved in the Softklone case, based their screen design on human factors analysis, according to counsel for the copyright proprietor.[2]

The Softklone court found that the arrangement of menu commands under particular parameter group headings "aids the user in easier understanding . . . of the commands." The court also commented on the use of highlighting and capitalizing the first two letters of commands listed in the main menu to indicate that the emphasized two letters were the keystrokes to invoke the commands. The court found that using these expedients "assist the user in knowing which symbols to enter to activate the various commands."[3] The court then held that these facts supported copyrightability of the menu screen display.

Yet, assisting the user in such "easier understanding" and in knowing what to do to activate the parts of a computer program is a goal of human factors analysis.

Generally accepted techniques and approaches to screen design should be considered the property of all; that is, they should be considered part of the public domain. These techniques are necessary instruments of technological advancement in the software field and of any project for proliferation of computer usage. Necessary techniques typically, although not invariably, cannot be identified as the creations of particular individuals. Instead these techniques seem to be the cumulative result of incremental contributions of a mass of anonymous workers in the field.[1] Hence, there is often an appearance of unfairness when any single person is permitted to assert proprietary rights over such a technique, or over the last incremental step in its development. The claim of unfairness seems especially justified when no prior determination of novelty or unusual technical advance on the part of the person asserting proprietary rights has occurred (for example, by a government patent office).

Software professionals frequently proclaim the unfairness of allowing assertions of proprietary rights in such techniques. (Witness the recent picketing of Lotus over its user interface lawsuits, and threats of boycotts against Lotus and Apple because of allegedly "egregious" copyright litigation.)

Furthermore, the consequence of permitting anyone to assert exclusive rights over a necessary technique is to hinder others from making their own technical contributions in the field. The barricade becomes more substantial the more necessary the technique is to the work of others. The purpose of any legal regime of exclusive rights over technology would have to be to promote the progress of knowledge, technology, and human welfare. Hence, there are always limits to what the law will protect under any intellectual property system. Also, there is a point at which protection should be denied because it hinders rather than promotes the realization of those goals of progress. These general principles apply to screen display technology.

Human factors analysis

In broad terms, the generally accepted techniques and approaches to screen design that should be considered part of the public domain usually involve the application to screen design of human factors analysis. (Human factors analysis is the study of human interaction with computers, which may be based on logic or empirical data.) Thus, it is known that material on a menu or similar screen display should be ordered to group together functionally similar commands. The menu should vertically order commands by their frequency of use (so that the user may more easily find the most frequently used commands at the top of the menu). The menu should otherwise facilitate the user's task flow. Titles should be centered at the top of a screen; command fields should be located at the bottom.

Those ideas are not proprietary, and the particular means of carrying out the ideas are usually functional and utilitarian. Only a few ways exist to implement these prescriptions. For example, only one way exists to center a caption at the top of a screen. For other examples of generally accepted screen design practice, see the box on this page.

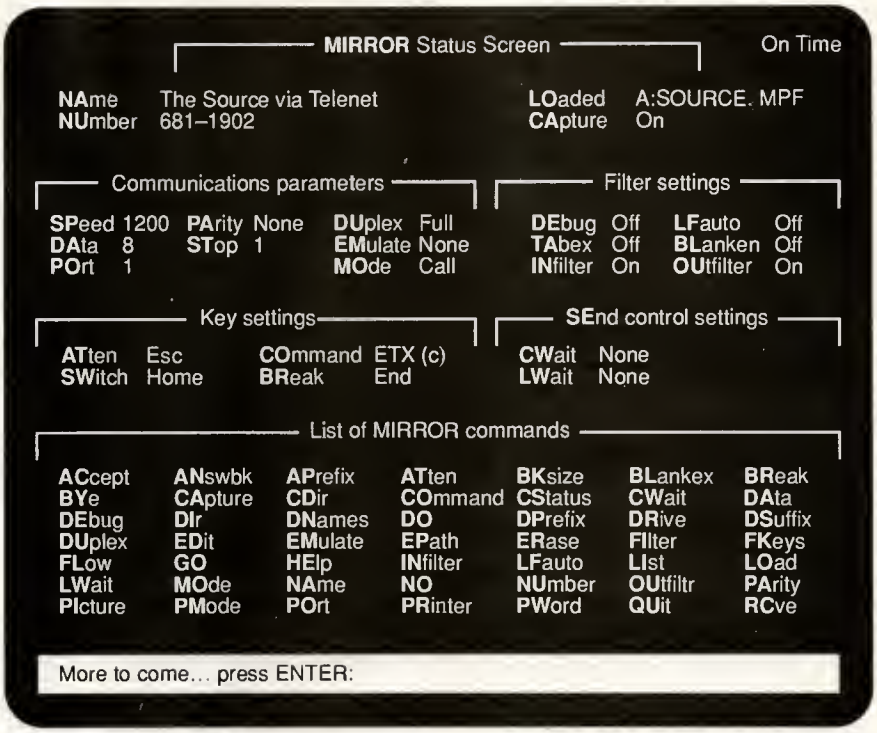
Example

When a particular technique of screen display is highly utilitarian, it is a mistake to hold a competitive screen to be a copyright infringement merely because its designer used the same utilitarian technique as an earlier market entrant did. To be sure, a great many "good design" ways may exist to arrange a screen display of some type, as shown by consumer acceptance in the marketplace of other computer program products or by other probative evidence. When such evidence occurs, any concern over the effects of copyright covering one such way would be misguided. But to the extent, if any, that copyright can preempt an important screen-design function, there are legitimate grounds for concern about overextending copyright protection of screens.

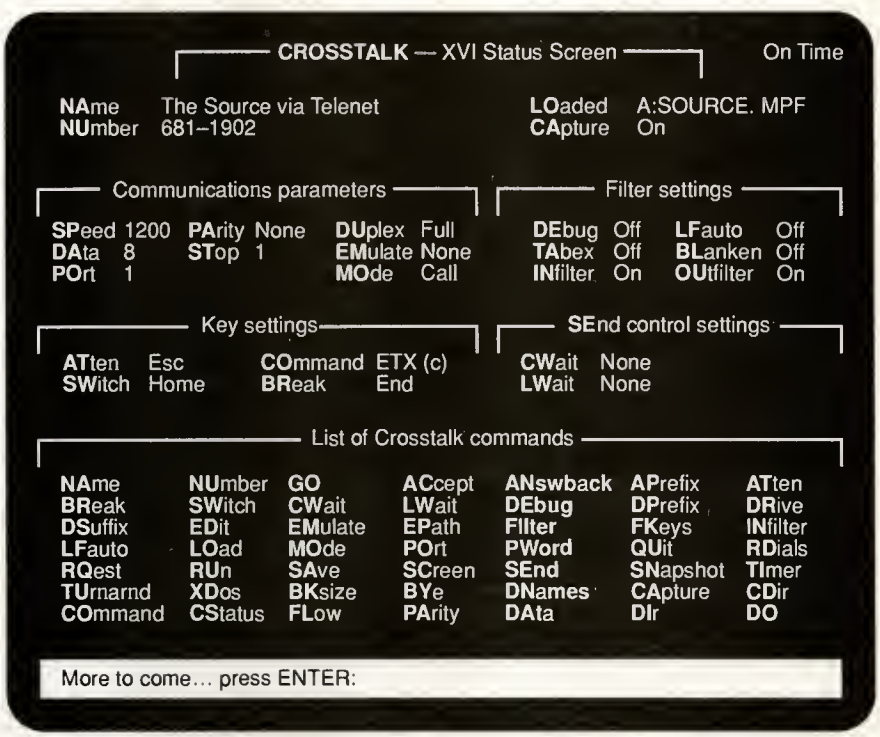
An example based on recent litigation illustrates the concerns that some have about the risk of preemption of utilitarian aspects of screens. In some programs (command-driven programs), a great many commands and parameters must be presented on a menu screen. Probably only a few acceptable techniques exist for doing this without producing a cluttered, unreadable, exasperating screen.

For example, consider a case in which 50 or more commands or parameters must be displayed at one time on a screen. (A screen would typically hold a maximum of 24 lines of 80 characters.) Designers often choose to place the first one or two letters of each command or parameter word in high-intensity video or inverse-video (highlighting). They also capitalize the same letter(s). Thus the designer would place **P**rint in a menu screen to mean the user should enter <P> to print the document, as in the menu of Figure 1. Another choice would place **QU**it in the menu to mean the user should enter <QU> to quit the program. Such use of highlighting for the first one or two letters of a command or parameter may be the only way to provide an uncluttered, readable screen. The technique is highly utilitarian in such contexts.

The accompanying Figure 2 illustrates this type of menu; it is the main menu for the Crosstalk XVI communications program.[4] This figure shows the pair of main menu screen displays involved in *Digital Communications Associates, Inc.* v. *Softklone Distributing Corp.*[3] The Crosstalk XVI menu comes from the first software that appeared on the market. The Mirror menu appeared in a subsequent competitive product that Softklone marketed at a lower price. (For a product review of Mirror, see Hannum.[5])



(a)



(b)

Figure 2. Status screens of (a) Mirror and (b) Crosstalk XVI programs.

The defendant's unauthorized imitation of the plaintiff's use of the highlighting technique in the Crosstalk program was a principal factor that led to a finding of copyright infringement in the Softklone case.

The court's abstract formulation of the applicable legal theory appears correct at an abstract level. That is, the

court recognized that if a screen display technique is necessary for a particular purpose, copyright law will not prevent others from using the technique. The court found that a vast number of other techniques besides highlighting could be used to design the screen display for this type of menu. The court therefore felt that interpreting the plaintiff's copyright in the Crosstalk screen display to cover this use of highlighting would not preempt the utilitarian. Accordingly, the court concluded that the defendant's appropriation of the technique was copyright infringement.

But further examination shows that the court either misunderstood the situation or was not properly briefed about how to design screens. The court listed in its opinion many other techniques that were allegedly useful substitutes for highlighting in a command-driven menu. But, all of them would produce cluttered, irritating, user-unfriendly screens. The court appears to have preempted a necessary screen-design technique by its ruling, even though it attempted to follow a correct principle. (Further discussion appears in the adjacent box.)

The outcome of this litigation thus illustrates the risks that legal protection of screen displays can pose to screen designers, entrepreneurs employing them, and investors in screen display technology. A short answer to that objection can be stated. Fact-finding mistakes can always be made, appellate courts exist to correct plain error, and screen designers accused of infringement should hire counsel who will present their cases properly. That short answer does not eliminate concern, however, by those who fear that perennially underfunded start-up entrepreneurs too often will be stifled by the threat of litigation. Those observers may also fear that plain error will too often occur when such controversies are presented for resolution to judges who are not technically sophisticated.

At the very least, those who fear these things may assert that this kind of litigation risk calls for a persuasive cost-benefit analysis justifying legal protection of screen displays. The result of such protection may be, whether by actual court decision or by the effect of intimidation, to preempt necessary, utilitarian aspects of screen design. The end result may be to hinder technological advance in this field.

Conventional techniques, standards

The problem of overprotecting utilitarian features of screen displays is broader than just the preemption of essential or necessary techniques. A screen design device may be merely conventional and useful, rather than necessary in the sense that human factors analysis or software engineering practice dictated its initial adoption. Yet, risk of its preemption may still be cause for concern. The distinction between convention and necessity is not always clear. The same technique may be one or the other (or both) depending on the factual context. For example, highlighting the first one or two letters of a command or parameter, and then using the highlighted alphanumerics to represent the keystrokes for invoking those commands, is sometimes so highly utilitarian that it is necessary. (See Figure 2, the example given earlier of a very crowded screen for a command-driven program.) Sometimes, highlighting is just a convenient, albeit inessential, convention. An example of that, perhaps, is the relatively uncrowded screen of the hypothetical word processing menu illustrated in Figure 1.

Convention is utilitarian, for it facilitates and speeds communication of ideas. A conventional gesture, for example, may substitute for many words and more emphatically convey an idea. Conventions occur in the traditional subject matter of copyright, such as plays, films, and comic strips. For example, consider the use of a light bulb in a balloon over a comic strip character's head to mean the dawning of an idea and the use of stars to mean a sensation of pain are economical means for communicating ideas. So, too, is use of "#%$@" to mean deleted expletives.

Thus, preempting conventional features of screen designs would impose additional production costs on screen designers and additional learning costs on users. While the effect is not as drastic as that of preempting necessary techniques, nonetheless it deserves consideration. Hence, if highlighting initial letters in a word is a useful, conventional way to identify keystrokes for commands, interpreting copyright law to protect the first user of that technique in a particular type of application may unduly hinder the efforts of others. Granting legal protection may be objec-

continued on p. 92

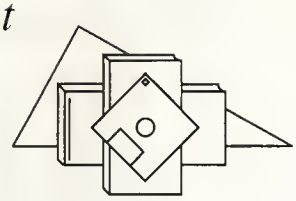
The Softklone Court vs. Screen-Design Human Factors Analysis

The Softklone court[6] made an assumption at trial that the main menu must be huge (approximately 100 choices were judged by both plaintiff and defendant as the customers' demand). That assumption is crucial to the present legal analysis.

If the contrary factual premise is adopted, namely, that the main menu can be broken up into four to six menus, the cluttered screen problem that dominates the legal analysis would probably (or should) evaporate. (In fact, eventually both parties released newer versions with the main menu broken into sets of smaller menus.) Furthermore, if a menu-driven mode of operating the computer program is adopted instead of a command-driven mode, the need to make particular keystrokes stand out on the screen would decrease or vanish.

Nonetheless, the court's opinion in the Softklone case defines the legal controversy in terms of a requirement of a huge, command-driven menu and presupposes a monochrome screen. Those assumptions lead to the conclusion drawn here that the court's ruling preempted one of the only two or three effective means of providing the necessary menu. (Perhaps, high-intensity video is the *only* available technique. Use of inverse video may be undesirable for several technical reasons.) Galitz[7] says that underlining is no good and that inverse video causes several problems. That leaves only high-intensity video as an option. See also what I said in *IEEE Micro*, which is similar.[8]

Department



Micro Review

Richard Mateosian
2919 Forest Avenue
Berkeley, CA 94705-1310
(415) 540-7745

A new Macintosh environment

Recently I acquired use of a Macintosh SE/30, an event that forced me to deal with issues I've been able to ignore with my old Mac Plus. For example, my old screen-saver program didn't seem to work with the new system software, so I started using Pyro, Fifth Generation's screen saver. Fortunately, the latest version of Pyro has an option that allows display of a moving analog clock, which was the only feature of my old screen saver that the old Pyro didn't have.

I had been using Suitcase to manage fonts and desk accessories (see my February 1988 column). Since the 1988 review, I have received review copies of Suitcase II and Font/DA Juggler Plus. I wasn't able to make the latter work properly (the Mac went into never-never land on start-up), but I don't know whose fault that was. Suitcase II worked well and has become an indispensable part of my system.

The new environment also stimulated me to try some of the larger new packages, like Mathematica and Wingz. I wasn't able to bring Mathematica up on my SE/30, probably because it has only two megabytes of main memory. I tried Mathematica on a Mac II CX with eight megabytes of main memory, and it came up and ran. It seems like a wonderful program, but I didn't have enough time on that system to exercise all of its functions. Wingz came up on the SE/30 and ran well. It also runs on my Mac Plus, but slowly. I still use Excel, but I might use Wingz for some purposes.

Suitcase II (Fifth Generation Systems, Inc., $79)

Suitcase II bills itself as "complete font and desk accessory liberation for your Apple Macintosh computer," and this is really true. Suitcase II works a lot like the original Suitcase, only better. No longer do suitcases to be opened on start-up need to be in special folders. Instead, on start-up Suitcase II simply opens any suitcases it had open the last time anything changed. In addition to fonts and desk accessories, it also handles F keys and sounds, which are also handled much more cleanly with the new Apple system software than they were in the older version that I had been using.

Suitcase comes with Pyro, the screen-saver program, and with two helpful utilities, Font Harmony and Font & Sound Valet. Font & Sound Valet allows suitcase files to be compacted, resulting in substantial space savings on disk with no appreciable performance penalty. Font Harmony resolves font-numbering conflicts and lets you combine style variations in font families into a single family. Thus, for example, rather than separate entries in your font menus for (Bookman, Bold Bookman, Italic Bookman, and so on), Font Harmony combines these into one family, Bookman, preserving any internal references that these fonts make to one another.

My only complaint about all of this is that using Font Harmony requires a tricky sequence of operations. It managed to hang up my system mysteriously several times, and I had to reboot. When the smoke cleared, however, I was able to reach the desired final result without destruction of any of the files that I had been working on when the crashes occurred.

Wingz (Informix, $395); and ***Mastering Wingz, The Official Introduction to Wingz Presentation Spreadsheet***, Fred E. Davis and Elna R. Tymes (Bantam, New York, 1989, 347 pp.; $24.95)

Wingz was touted long in advance of its arrival as the next generation of integrated presentation software. And, if the documentation is anything to judge by, one has to question whether it is really ready yet.

I began my review of Wingz by reading *Mastering Wingz*. This book, while separately published, contains the enthusiastic recommendation of Michael J. Brown, president of the Informix Workstation Products Division, who lauds the authors for their devotion to detail. Let it suffice to say that this is one of the worst edited and worst published books I have ever encountered and that at least some of the blame has to fall on the authors.

I don't mean to say that you shouldn't read *Mastering Wingz*. To the contrary, if you want to work with Wingz, this is a good introduction (after watching the instructional videotape that usually comes bundled with the program). The vast majority of the book's many errors are typographical, spelling, and grammatical flaws, which most readers can get past without difficulty. The introduction to the features of Wingz is well organized and comprehensive; I found only a few areas where the explanations are wrong, confusing, or misleading. The worst problems of this sort are in the chapter on formulas and functions and the chapter on programming in Hyper Script. Most of these problems could have been avoided if the book had been

0272-1732/89/0800-0011$01.00 © 1989 IEEE

read by an editor who understood its contents.

Having started with *Mastering Wingz*, I did not spend much time with the documentation provided by Informix. I did, however, refer to the Informix documentation on several occasions when I simply could not understand what the authors of *Mastering Wingz* were trying to tell me. In some of those instances I found the same gibberish, almost verbatim, in the Informix documentation.

After reading the book, I tried playing with the program. Over the course of a month or so, I ran it on my Mac Plus, an SE/30, and a II CX. It runs much faster on the 68020/68881-based machines than on my 68000-based Mac Plus. It seemed to handle ordinary spreadsheet functions more or less instantaneously on my Mac Plus, but its fancy graphics were painfully slow on that machine. On my Mac Plus I was able to write out in SYSLK format the Excel spreadsheet I use for my cash planning and read it into Wingz. It seemed to work perfectly, and I was quickly able to construct a helpful graphic representation of the row of data representing my cash balances for each calendar period. However, I did not care for the look and feel of Wingz nearly as much as that of Excel.

The jury is still out on Wingz, but its graphics features are definitely easier to use than those of my version of Excel (1.5).

This and That

Unix in a Nutshell handbooks (O'Reilly & Associates, Inc.)

O'Reilly sent me an assortment of their Unix reference books. They publish works with down-to-earth titles like *Reading and Writing Termcap Entries* (87 pages) and *Programming with Curses*. The first one that I looked at was *Managing Projects with Make* by Steve Talbott (1987, 77 pp., $9). Since I have used the Make utility to manage both programming and documentation projects, under both Unix and MS-DOS, I thought that would be a good place to start.

Make is a program that manages file dependencies. A typical programming project builds one or more program files by linking a variety of libraries and object files. The libraries may be updated periodically, and the object files are produced from source files by assembly or compilation. Make solves the problem of keeping track of all of the interfile dependencies and assuring that the program files are kept up to date without unnecessary operations. It does this by using a makefile, a central repository of information about file dependencies and of rules for creating each target and intermediate file.

The problem that this book attacks is the paucity of clear documentation of Make. Make's syntax takes a little getting used to, and Make's suffix rules, a set of shortcuts for specifying predictable dependencies, lead to cryptic makefiles. In addition, Make allows macro definitions using shell variables and also allows setting of some variable values from the command line. Furthermore, Make sometimes imitates the operation of a shell and sometimes passes commands to a shell for execution. In the latter case, each line is given to a separate shell, so that the effect of a sequence of commands can depend upon whether the commands are written on one line or on several. None of these features is presented tutorially in the Unix manuals.

The book gets off to a bad start. Its first example is a 10-line makefile that has been transformed into a 12-line file because the first two lines are too long to fit the column width of the book's text. The lines are numbered from one through 12 on the page, then referred to in the descriptive text by numbers appropriate to the 10-line version. This renumbering causes only a little confusion to someone familiar with the material, but presumably the target audience consists of people unfamiliar with the material. I have some theories about how this might have happened. It seems to be the result of partial automation of the publication process.

After this early confusion, however, the book improves. It leads the reader through a lot of nitty-gritty detail and may live up to the advertisement on the cover (I haven't seen the competition): ". . . without question the clearest description of Make ever written."

***American Men and Women of Science*, 17th ed., 1989-90** (Bowker, 1989, 8 volumes, $650)

This is an enormous compilation of curricula vitae of scientists. Over 7,300 pages of short entries summarize the careers of approximately 125,000 men and women. A 490-page Discipline Index groups the names into 160 fields of activity, using categories established by the National Science Foundation.

Unfortunately, the coverage of computer science is relatively scanty. For example, the editors devote only 11 pages of the index to computer science. These cover six categories: general (5.8 pages), hardware systems (0.7 page), intelligent systems (1 page), software systems (2.4 pages), theory (0.5 page), and other (0.6 page).

As Bowker points out in the press release it sent me, well over four million scientists and engineers work in the US and Canada, so some selection is necessary. I am sure that they apply some worthy criteria to that selection. On the other hand, I suspect that there is a degree of randomness as well. Bowker includes only two members of the *IEEE Micro* editorial board (Kahaner and Mickle), and in neither case is their connection with *Micro* mentioned.

I have also checked the attendance list (approximately 85 people) from a microcomputer workshop I recently attended, finding only about five of them in this reference work. Among those omitted was someone whose name is on the patent for the first microprocessor.

I can't assess the accuracy of the information included. I didn't see any glaring errors, but most of the entries I read were for people whom I don't know. I don't recommend that you run out and buy this set, but if you have a well-endowed library at your institution, they might consider investing in it. If you need the kind of information it contains, about someone who happens to be listed, this is as good a way to get it as any I can think of.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 177 **Medium** 178 **High** 179

High-Performance Microprocessors: The RISC Dilemma

Victor K.L. Huang
AT&T Bell Laboratories

Many systems vendors and microprocessor users find themselves forced into making a major decision: Should they invest in the current RISC upsurge, or should they ignore it?

Since the emergence of the first commercial reduced instruction-set computing processor, the Fairchild (now Intergraph) Clipper, our industry has witnessed increasingly sophisticated technological advances. Advances in silicon and processing technologies now give us clock speeds in excess of 40 MHz and chips with 1 million transistors! The quest for high performance and increased functionality, not just raw speed, led to sophisticated architectural techniques, specifically three- to four-stage pipelining and large instruction and data caches. As a result, we can achieve single-cycle execution for all instructions, one of the traditional RISC principles. Added functionality, such as floating-point units and graphical processors, also contributed to the high-performance race.

Paralleling these explosive technological advances is an equally important milestone: the development and growth of microprocessor software and peripheral-support infrastructure. In essence, the microprocessor industry has matured! From the first 4-bit processor produced in 1971 by Intel Corp., the technology has evolved and contributed significantly to our Information Age. We have effectively unleashed the power of micro-mainframes on desktop machines, proliferating distributed and networked computing via local area networking, personal computers, and workstations.

To retain and increase market share, leading CISC (complex instruction-set computers) vendors, such as Intel and Motorola, concentrated their strategic efforts on nurturing their own infrastructure of software development, peripheral support, bus standards, and single-board and subsystems vendors. These efforts resulted in substantial embedded bases of users. Software and applications have in fact forced de facto standards on the industry, notably MS-DOS in the PC market dominated by the Intel 80x86 series.

Users, when selecting microprocessors, find themselves faced with the RISC option. If they choose a RISC for high-performance computing, they might be unable to take advantage of the embedded applications-software base due to compatibility and porting issues. If they take the more conservative hybrid path (complex reduced instruction-set processor), they could lead themselves into technological obsolescence should the future prove to be dominated by RISCs.

Observing lessons learnt from the CISC evolution trends, users understand that having a large applications software base and forming strategic alliances with innovative and large systems houses are crucial to the success and survival of RISC vendors. However, they also realize that a major constraint exists: Systems houses are reluctant to commit to a RISC platform without users to develop more software, while users shy away from new systems unless there is a sufficient software base. This Catch-22 situation has created a new playing field in the microprocessor industry. Now, RISC silicon and software vendors compete with the established CISC/RISC houses.

As it turns out, this Catch 22 can be resolved. The systems houses' willingness to adapt and migrate to a new software platform seems to depend on two factors. RISCs offer substantially higher performance over CISCs, and a standard software/operating systems environment cuts across architectural boundaries. Such an opportunity was aggressively pursued by a leading workstation vendor, for example. This vendor advocated a high-performance, open software-standard platform via RISC (for performance) and Unix (for an open systems environment).

Now, industry pundits seem to project RISC as the wave of the future. Their predictions depend on three industry trends converging to a true open systems environment: high-performance computing through RISCs, semiconductor and circuit design advances allowing complete systems on a chip, and Unix—the operating system of choice of all RISC processors.

Events in the industry seem to bear this prediction out. In establishing open systems and standards, large Unix consortiums (Open Software Foundation, Unix International, X/Open) and strategic alliances between systems houses and RISC vendors (Sparc/Toshiba, DEC/MIPS) have formed. Following this trend, consortiums designed to propagate RISC processors as open standards are also cropping up (Sparc International, 88open). These consortiums encourage rapid applications and software development, establish applications binary interfaces (ABIs), and ensure compatibility and portability.

Industry observers believe a shakeout is imminent, with the RISC technology undoubtedly following its

0272-1732/89/0800-0013$01.00 © 1989 IEEE

Guest Editor's Introduction

CISC predecessors in its technological evolution converging on main market vendors. But stakes are high: Observers fully expect RISCs to dominate CISCs in the workstation market in the 1990s.[1] This prognosis is further reinforced by the actions of the market leaders, Intel and Motorola. Both companies have announced RISC products. Other strong contenders are Sun Microsystems Inc. with its Sparc architecture (produced by Bipolar Integrated Technology, LSI Logic, Cypress, Fujitsu, and Texas Instruments) and the MIPS Computer Systems' architecture (produced by LSI Logic, Integrated Device Technology, Performance Semiconductor, NEC, and Siemens). In terms of volume, though, the Intergraph Clipper still leads the market with volume production in 1988 that garnered impressive design wins based on availability.

Current technological breakthroughs contribute in two ways: speed and density. At a recent PC Expo (May 1989), Intel Vice President David House predicted that we will see 100 million transistors on a chip by the year 2000. House claimed we would see these chips offering 60-MHz clock speeds, four CPUs, and a digital video interactive processing unit.[2] This possibility would definitely allow CISC vendors the capabilities to provide RISC features, blurring the distinction between the two. It would also enable them to remain compatible with their embedded base. Don't count the CISC vendors out yet!!

The speed breakthroughs are just as impressive. Bipolar ECL (emmitter-coupled logic) and BiCMOS (CMOS input transistors and bipolar output transistors) technologies resulted in bipolar RISC processors (newly nicknamed BRISCs). These processors push clock rates upwards of 100 MHz and offer a projected throughput of 200 million instructions per second by 1993.[3]

All of the major players participate in the speed advances. They've used bipolar ECL as the technology of choice to produce the Sparc (by BIT), the 88000 (by Motorola and Data General), and the MIPS (by NEC). In the BiCMOS version, Cypress and Fujitsu backed the Sparc, and IDT produced the MIPS. Fujitsu also plans a bipolar Clipper. Interestingly enough, these high-performance RISC machines are not targeted at present CMOS RISC markets, but at high-end workstations, minicomputers, and superminis. Trends at this level of applications include open architectures and standardized operating systems such as Unix. And, these developers are willing to shift from traditional CISC machines for substantive performance enhancement and scale of integration—both objectives that bipolar RISC vendors are intent on fulfilling.

These are but some of the issues facing many vendors and users in the strategic positioning of their products and influencing their processor-selection process. With this background, you will find that this special issue on high-performance processors brings you a snapshot of some of the latest in today's available technology. This issue features Intel's entry in the RISC race. Les Kohn and Neal Margulis discuss the i860 processor in detail. Motorola's RISC entry, the 88000 processor, was featured in *IEEE Micro*'s April issue; now we offer their MC68332, a high-performance microcontroller for real-time control applications. Joe Jelemensky and crew discuss how VLSI (very large scale integration) technology brings high-performance microprocessor capability to meet the specific needs of a specific industry (real-time control).

In addition, we are indeed fortunate to present in this special issue a discussion of architectural feature comparisons of three RISC processors. Rich Piepho and Bill Wu of AT&T discuss the i860, 88000, and Sparc architectures. This article is especially timely today because it points out the merits of different architectures so readers can increase their knowledge of this newest technology.

We hope the articles presented here will provide our readers with better guidance and renewed awareness of the high-performance processors. I have particularly enjoyed guest editing this issue as we are right in the middle of this relentless march in technology. In fact, the frequency with which advancements are moving warrants another high-performance issue very soon.

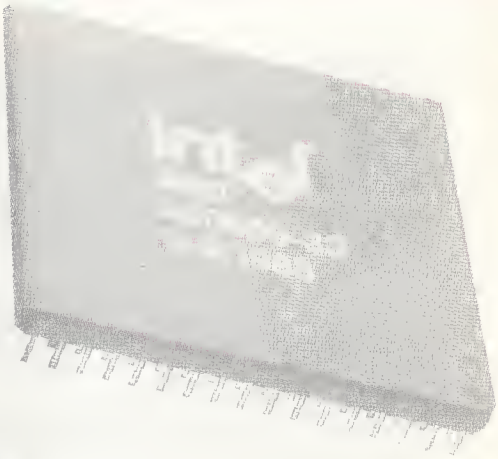
References

1. Jonah McLeod, "Tough Choices Ahead for Microprocessors," *Electronics*, May 1989, pp. 10-78.
2. *Electronic Eng. Times*, "What Do You Do With 100 Million Transistors?" June 26, 1989, Vol. 544, p.1.
3. Bernard C. Cole, "CPUs Are Marching to a BRISC Beat," *Electronics*, June 1989, pp. 110-113.

Victor K.L. Huang has supervised the design and development of the WE32200 CPU at AT&T Bell Laboratories. His responsibilities have also included the design of 4-, 8-, and 32-bit microprocessors.

Huang received a BSEE from the Virginia Military Institute and an MSEE and PhD from the University of Virginia. He is a senior member of the IEEE and a member of the IEEE Computer Society, the Industrial Electronics Society, and the editorial board of *IEEE Micro*.

Questions concerning this issue may be directed to Victor K.L. Huang, AT&T Bell Laboratories, Room 2K-303, Crawford Corners Road, Holmdel, NJ 07733.



Introducing the Intel i860 64-Bit Microprocessor

A million-transistor budget helps this RISC deliver balanced MIPS, Mflops, and graphics performance with no data bottlenecks.

Les Kohn
Neal Margulis

Intel Corp.

The single-chip i860 CPU—a 64-bit, RISC-based microprocessor—executes parallel instructions using mainframe and supercomputer architectural concepts. We designed the 1,000,000-transistor, 10 mm × 15 mm processor (see Figure 1 on the next page) for balanced integer, floating-point, and graphics performance, using the company's latest generation CAD tools and 1-micrometer semiconductor process.

To accommodate our performance goals, we divided the chip area evenly between blocks for integer operations, floating-point operations, and instruction and data cache memories. Inclusion of the RISC (reduced instruction set computing) core, floating-point units, and caches on one chip lets us design wider internal buses, eliminate interchip communication overhead, and offer higher performance. As a result, the i860 avoids off-chip delays and allows users to scale the clock beyond the current 33- and 40-MHz speeds.

We designed the i860 for performance-driven applications such as workstations, minicomputers, application accelerators for existing processors, and parallel supercomputers. The i860 CPU design began with the specification of a general-purpose RISC integer core. However, we felt it necessary to go beyond the traditional 32-bit, one-instruction-per-clock RISC processor. A 64-bit architecture provides the data and instruction bandwidth needed to support multiple operations in each clock cycle. The balanced performance between integer and floating-point computations produces the raw computing power required to support demanding applications such as modeling and simulations.

Finally, we recognized a synergistic opportunity to incorporate a 3D graphics unit that supports interactive visualization of results. The architecture of the i860 CPU provides a complete platform for software vendors developing i860 applications.

Architecture overview. The i860 CPU includes the following units on one chip (see Figure 2):

- the RISC integer core,
- a memory management unit with paging,
- a floating-point control unit,
- a floating-point adder unit,
- a floating-point multiplier unit,
- a 3D graphics unit,

0272-1732/89/0800-0015$01.0 © 1989 IEEE

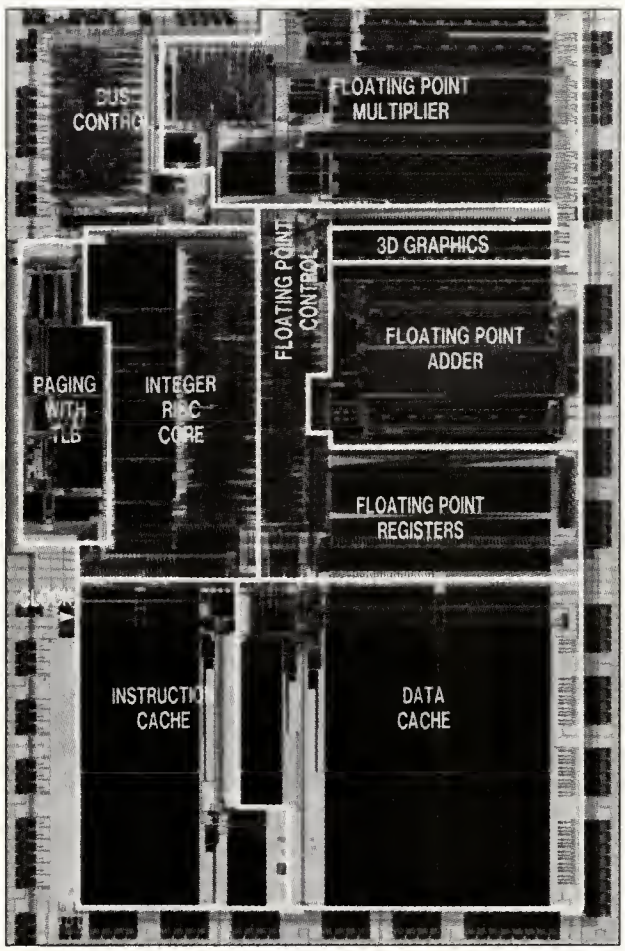


Figure 1. Die photograph of the i860 CPU.

- a 4-Kbyte instruction cache,
- an 8-Kbyte data cache, and
- a bus control unit.

Parallel execution. To support the performance available from multiple functional units, the i860 CPU issues up to three operations each clock cycle. In single-instruction mode, the processor issues either a RISC core instruction or a floating-point instruction each cycle. This mode is useful when the instruction performs scalar operations such as operating system routines.

In dual-instruction mode, the RISC core fetches two 32-bit instructions each clock cycle using the 64-bit-wide instruction cache. One 32-bit instruction moves to the RISC core, and the other moves to the floating-point section for parallel execution. This mode allows the RISC core to keep the floating-point units fed by fetching and storing information and performing loop control, while the floating-point section operates on the data.

The floating-point instructions include a set of operations that initiate both an add and a multiply. The add and multiply, combined with the integer operation, result in three operations each clock cycle. With this fine-grained parallelism, the architecture can support traditional vector processing by software libraries that implement a vector instruction set. The inner loops of the software vector routines operate up to the peak floating-point hardware rate of 80 million floating-point operations per second. Consistent with RISC philosophy, the i860 CPU achieves the performance of hardware vector instructions without the complex control logic of hardware vector instructions. The fine-grained parallelism can also be used in other parallel algorithms that cannot be vectorized.

Register and addressing model. The i860 microprocessor contains separate register files for the integer and floating-point units to support parallel execution. In addition to these register files, as can be seen in Figure 3 on page 18, are six control registers and four special-purpose registers. The RISC core contains the integer register file of thirty-two 32-bit registers, designated R0 through R31 and used for storing addresses or data. The floating-point control unit contains a separate set of thirty-two 32-bit floating-point registers designated F0 through F31. These registers can be addressed individually, as sixteen 64-bit registers, or as eight 128-bit registers. The integer registers contain three ports. Five ports in the floating-point registers allow them to be used as a data staging area for performing loads and stores in parallel with floating-point operations.

The i860 operates on standard integer and floating-point data, as well as pixel data formats for graphics operations. All operations on the integer registers execute on 32-bit data as signed or unsigned operations and additional add and subtract instructions that operate on 64-bit-long words. All 64-bit operations occur in the floating-point registers.

The i860 microprocessor supports a paged virtual address space of four gigabytes. Therefore, data and instructions can be stored anywhere in that space, and multibyte data values are addressed by specifying their lowest addressed byte. Data must be accessed on boundaries that are multiples of their size. For example, two-byte data must be aligned to an address divisible by two, four-byte data on an address divisible by four, and so on, up to 16-byte data values. Data in memory can be stored in either little-endian or big-endian format. (Little-endian format sends the least significant byte, D7-D0, first to the lowest memory address, while big-endian sends the most significant byte first.) Code is always stored in little-endian format. Support for big-endian data allows the processor to operate on data produced by a big-endian processor, without performing a lengthy data conversion.

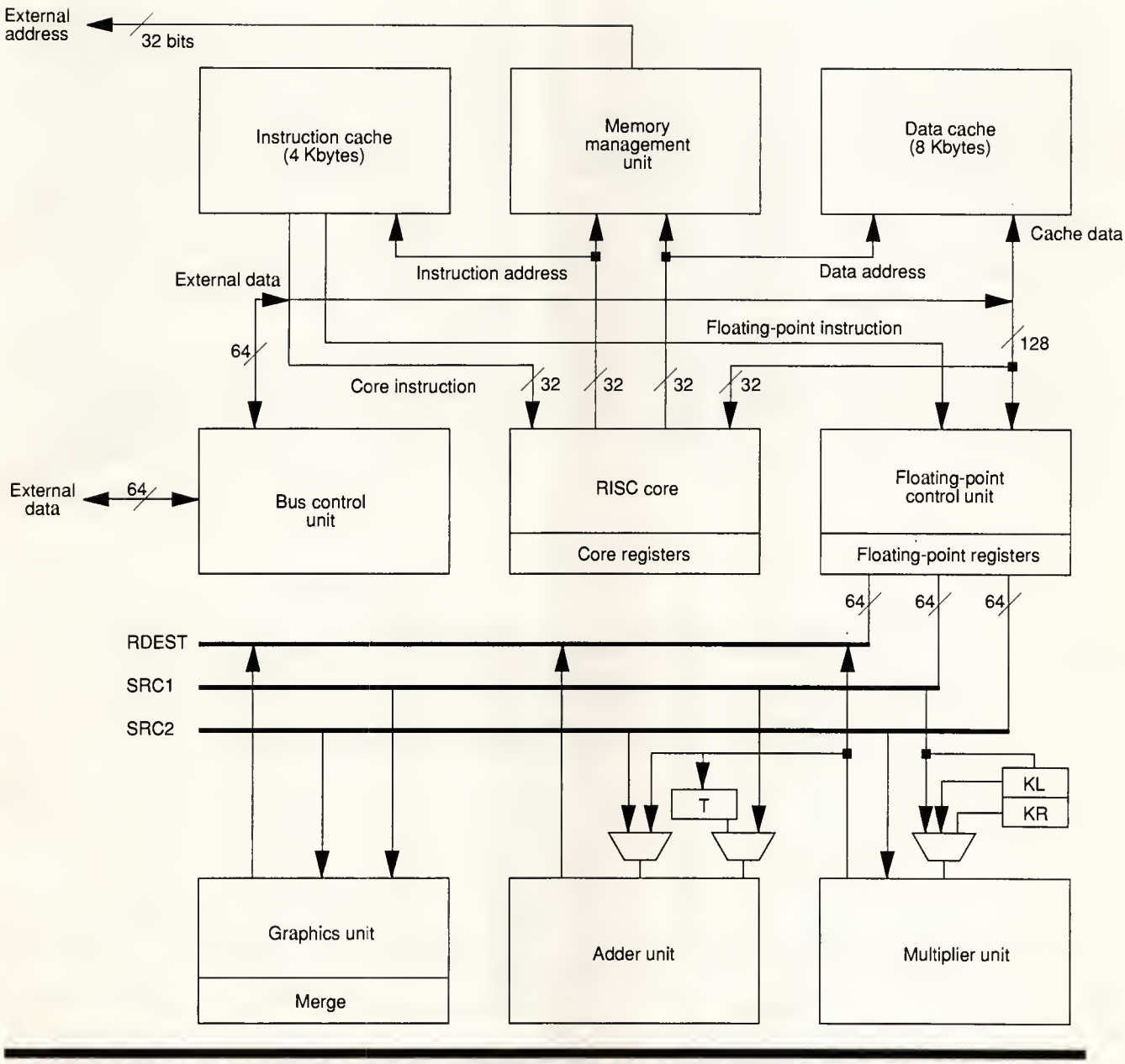


Figure 2. Functional units and data paths of the i860 microprocessor.

RISC core

The RISC core fetches both integer and floating-point instructions. It executes load, store, integer, bit, and control transfer instructions. Table 1 on page 19 lists the full instruction set with the 42 core unit instructions and their mnemonics in the left column. All instructions are 32 bits long and follow the load/store, three-operand style of traditional RISC designs. Only load and store instructions operate on memory; all other instructions operate on registers. Most instructions allow users to specify two source registers and a third register for storing the results.

A key feature of the core unit is its ability to execute most instructions in one clock cycle. The RISC core contains a pipeline consisting of four stages: fetch, decode, execute, and write. We used several techniques to hide clock cycles of instructions that may take more

time to complete. Integer register loads from memory take one execution cycle, and the next instruction can begin on the following cycle.

The processor uses a scoreboarding technique to guarantee proper operation of the code and allow the highest possible performance. The scoreboard keeps a history of which registers await data from memory. The actual loading of data takes one clock cycle if it is held in the cache memory buffer available for ready access, but several cycles if it is in main memory. Using scoreboarding, the i860 microprocessor continues execution unless a subsequent instruction attempts to use the data before it is loaded. This condition would cause execution to freeze. An optimizing compiler can organize the code so that freezing rarely occurs by not referencing the load data in the following cycle. Because the hardware implements scoreboarding, it is never necessary to insert NO-OP instructions.

We included several control flow optimizations in the core instruction set. The conditional branch instructions have variations with and without a delay slot. A delay slot allows the processor to execute an instruction following a branch while it is fetching from the branch target. Having both delayed and nondelayed variations of branch instructions allows the compiler to optimize the code easily, whether a branch is likely to be taken or not. Test and branch instructions execute in one clock cycle, a savings of one cycle when testing special cases. Finally, another one-cycle loop control instruction usefully handles tight loops, such as those in vector routines.

Instead of providing a limited set of locked operations, the RISC core provides lock and unlock instructions. With these two instructions a sequence of up to 32 instructions can be interlocked for multiprocessor synchronization. Thus, traditional test and set opera-

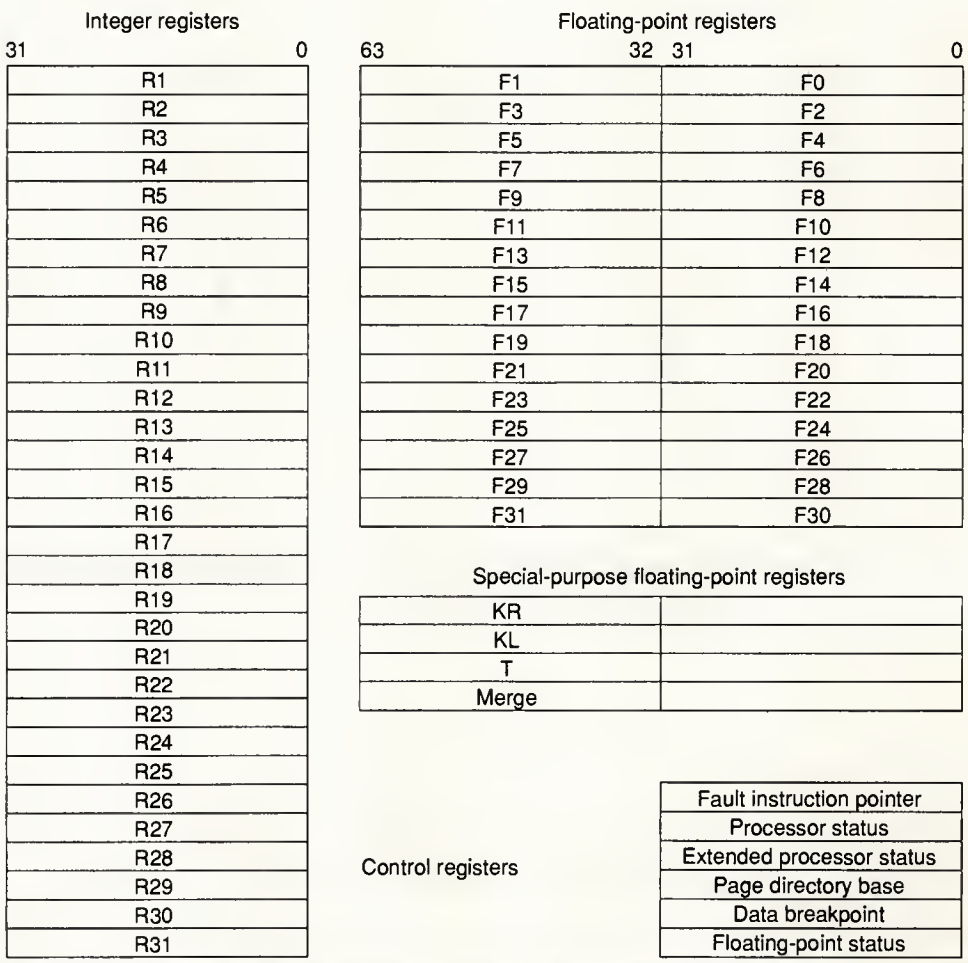


Figure 3. Register set.

Table 1.
Instruction-set summary.

Mnemonic	Description
Core unit	
Load and store instructions	
LD.X	Load integer
ST.X	Store integer
FLD.Y	F-P load
PFLD.Z	Pipelined F-P load
FST.Y	F-P store
PST.D	Pixel store
Register-to-register moves	
IXFR	Transfer integer to F-P register
FXFR	Transfer F-P to integer register
Integer arithmetic instructions	
ADDU	Add unsigned
ADDS	Add signed
SUBU	Subtract unsigned
SUBS	Subtract signed
Shift instructions	
SHL	Shift left
SHR	Shift right
SHRA	Shift right arithmetic
SHRD	Shift right double
Logical instructions	
AND	Logical AND
ANDH	Logical AND high
ANDNOT	Logical AND NOT
ANDNOTH	Logical AND NOT high
OR	Logical OR
ORH	Logical OR high
XOR	Logical exclusive OR
XORH	Logical exclusive OR high
Control-transfer instructions	
TRAP	Software trap
INTOVR	Software trap on integer overflow
BR	Branch direct
BRI	Branch indirect
BC	Branch on CC
BC.T	Branch on CC taken
BNC	Branch on not CC
BNC.T	Branch on not CC taken
BTE	Branch if equal
BTNE	Branch if not equal
BLA	Branch on LCC and add
CALL	Subroutine call
CALLI	Indirect subroutine call
System control instructions	
FLUSH	Cache flush
LD.C	Load from control register
ST.C	Store to control register
LOCK	Begin interlocked sequence
UNLOCK	End interlocked sequence

Mnemonic	Description
Floating-point unit	
Floating-point multiplier instructions	
FMUL.P	F-P multiply
PFMUL.P	Pipelined F-P multiply
PFMUL3.DD	Three-stage pipelined F-P multiply
FMLOW.P	F-P multiply low
FRCP.P	F-P reciprocal
FRSQR.P	F-P reciprocal square root
Floating-point adder instructions	
FADD.P	F-P add
PFADD.P	Pipelined F-P add
FSUB.P	F-P subtract
PFSUB.P	Pipelined F-P subtract
PFGT.P	Pipelined F-P greater-than compare
PFEQ.P	Pipelined F-P equal compare
FIX.P	F-P to integer conversion
PFIX.P	Pipelined F-P to integer conversion
FTRUNC.P	F-P to integer truncation
PFTRUNC.P	Pipelined F-P to integer truncation
PFLE.P	Pipelined F-P less than or equal
PAMOV	F-P adder move
PFAMOV	Pipelined F-P adder move
Dual-operation instructions	
PFAM.P	Pipelined F-P add and multiply
PFSM.P	Pipelined F-P subtract and multiply
PFMAM	Pipelined F-P multiply with add
PFMSM	Pipelined F-P multiply with subtract
Long integer instructions	
FLSUB.Z	Long-integer subtract
PFLSUB.Z	Pipelined long-integer subtract
FLADD.Z	Long-integer add
PFLADD.Z	Pipelined long-integer add
Graphics instructions	
FZCHKS	16-bit *z*-buffer check
PFZCHKS	Pipelined 16-bit *z*-buffer check
FZCHLD	32-bit *z*-buffer check
PFZCHLD	Pipelined 32-bit *z*-buffer check
FADDP	Add with pixel merge
PFADDP	Pipelined add with pixel merge
FADDZ	Add with *z* merge
PFADDZ	Pipelined add with *z* merge
FORM	OR with merge register
PFORM	Pipelined OR with merge register
Assembler pseudo-operations	
MOV	Integer register-register move
FMOV.Q	F-P register-register move
PFMOV.Q	Pipelined F-P register-register move
NOP	Core no-operation
FNOP	F-P no-operation

CC Condition code
F-P Floating-point
LCC Load condition code

tions as well as more sophisticated operations, such as compare and swap, can be performed.

The RISC core also executes a pixel store instruction. This instruction operates in conjunction with the graphics unit to eliminate hidden surfaces. Other instructions transfer integer and floating-point registers, examine and modify the control registers, and flush the data cache.

The six control registers accessible by core instructions are the

- PSR (processor status),
- EPSR (extended processor status),
- DB (data breakpoint),
- FIR (fault instruction),
- Dirbase (directory base), and
- FSR (floating-point status) registers.

The PSR contains state information relevant to the current process, such as trap-related and pixel information. The EPSR contains additional state information for the current process and information such as the processor type, stepping, and cache size. The DB register generates data breakpoints when the breakpoint is enabled and the address matched. The FIR stores the address of the instruction that causes a trap. The Dirbase register contains the control information for caching, address translation, and bus options. Finally, the FSR contains the floating-point trap and rounding-mode status for the current process. The four special-purpose registers are used with the dual-operation floating-point instructions (described later).

The core unit executes all loads and stores, including those to the floating-point registers. Two types of floating-point loads are available: FLD (floating-point load) and PFLD (pipelined floating-point load). The FLD instruction loads the floating-point register from the cache, or loads the data from memory and fills the cache line if the data is not in the cache. Up to four floating-point registers can be loaded from the cache in one clock cycle. This ability to perform 128-bit loads or stores in one clock cycle is crucial to supplying the data at the rate needed to keep the floating-point units executing. The FLD instruction processes scalar floating-point routines, vector data that can fit entirely in the cache, or sections of large data structures that are going to be reused.

For accessing data structures too large to fit into the on-chip cache, the core uses the PFLD instruction. The pipelined load places data directly into the floating-point registers without placing it in the data cache on a cache miss. This operation avoids displacing the data already in the cache that will be reused. Similarly on a store miss, the data writes through to memory without allocating a cache block. Thus, we avoid data cache thrashing, a crucial factor in achieving high sustained performance in large vector calculations.

PFLD also allows up to three accesses to be issued on the pipelined external bus before the data from the first cache miss is returned. The pipelined loads occur directly from memory and do not cause extra bus cycles to fill the cache line, avoiding bus accesses to data that is not needed. The full bus bandwidth of the external bus can be used even though cache misses are being processed. Autoincrement addressing, with an arbitrary increment, increases the flexibility and performance for accessing data structures.

Memory management

The i860's on-chip memory management unit implements the basic features needed for paged virtual memory management and page-level protection. We intentionally duplicated the memory management technique in the 386 and 486 microprocessors' paging system. In this way we can be sure that the processors easily exist in a common operating environment. The similar MMUs are also useful for reusing paging and virtual memory software that is written in C.

The address translation process maps virtual address space onto actual address space in fixed-size blocks called pages. While paging is enabled, the processor translates a linear address to a physical address using page tables. As used in mainframes, the i860 CPU page tables are arranged in a two-level hierarchy. (See Figure 4.) The directory table base (DTB), which is part of the Dirbase register, points to the page directory. This one-page-long directory contains address entries for 1,024 page tables. The page tables are also one page long, and their entries describe 1,024 pages. Each page is 4 Kbytes in size.

Figure 4 also shows the translation from a virtual address to a physical address. The processor uses the upper 10 bits of the linear address as an index into the directory. Each directory entry contains 20 bits of addressing information, part of which contains the address of a page table. The processor uses these 20 bits and the middle 10 bits of the linear address to form the page table address. The address contents of the page table entry and the lower 12 bits (nine address bits and the byte enables) of the linear address form the 32-bit physical address.

The processor creates the paging tables and stores them in memory when it creates the process. If the processor had to access these page tables in memory each time that a reference was made, performance would suffer greatly. To save the overhead of the page table lookups, the processor automatically caches mapping information for the 64 recently used pages in an on-chip, four-way, set-associative translation lookaside buffer. The TLB's 64 entries cover 4 Kbytes, each providing a total cover of 256 Kbytes of memory addresses. The TLB can be flushed by setting a bit in the Dirbase register.

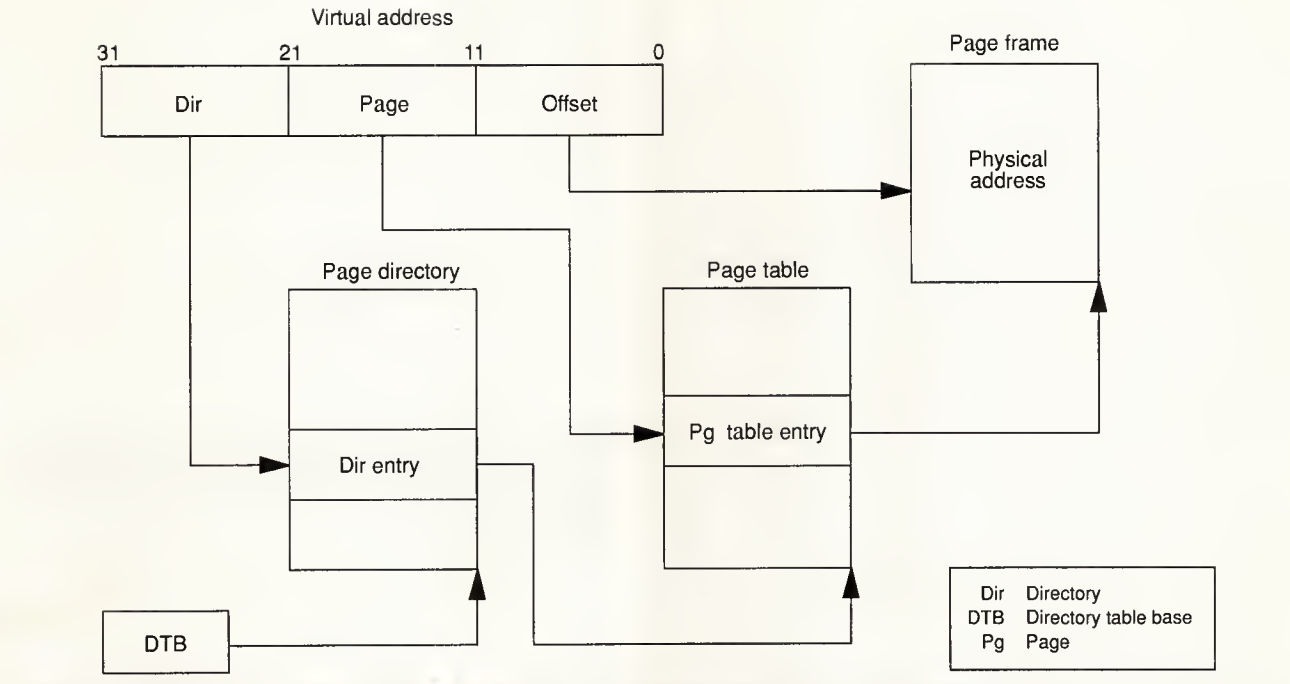


Figure 4. Virtual-to-physical address translation.

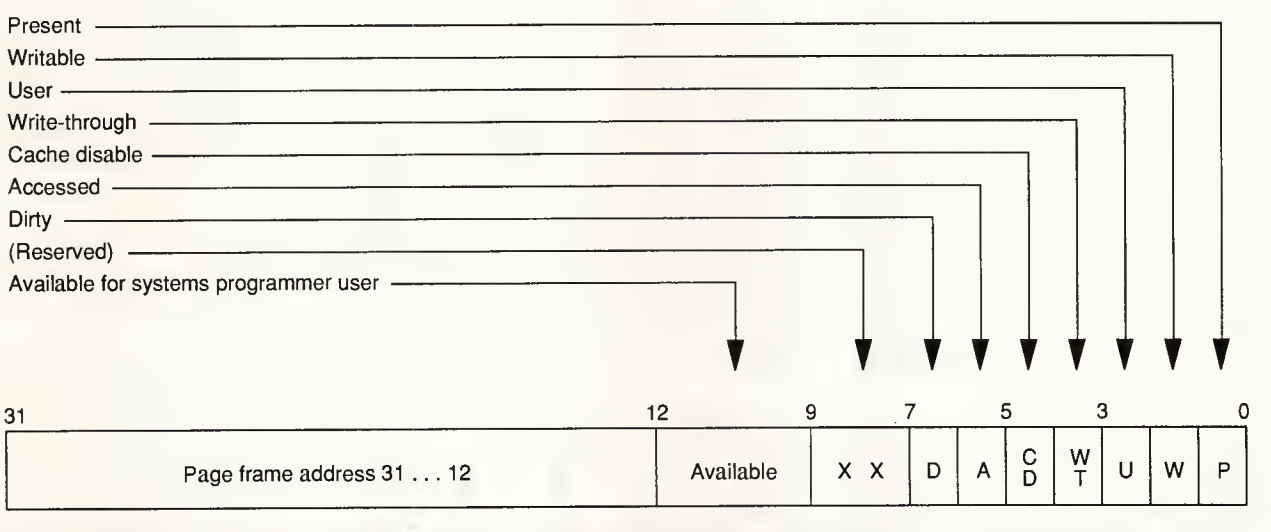


Figure 5. Format of a page table entry. (X indicates Intel reserved; do not use.)

Only when the processor does not find the mapping information for a page in the TLB does it perform a page table lookup from information stored in memory. When a TLB miss does occur, the processor performs the TLB entry replacement entirely in hardware. The hardware reads the virtual-to-physical mapping information from the page directory and the page table entries, and caches this information in the TLB.

The format of a page table entry can be seen in Figure 5. Paging protects supervisor memory from user accesses and also permits write protection of pages. The U (user) and W (write) bits control the access rights. The operating system can allow a user program to have read and write, read-only, or no access to a given page or page group. If a memory access violates the page protection attributes, such as U-level code writing a

read-only page, the system generates an exception. While at the user level, the system ignores store control instructions to certain control registers.

The U bit of the PSR is set to 0 when executing at the supervisor level, in which all present pages are readable. Normally, at this level, all pages are also writable. To support a memory management optimization called copy-on-write, the processor sets the write-protection (WP) bit of the EPSR. With WP set, any write to a page whose W bit is not set causes a trap, allowing an operating system to share pages between tasks without making a new copy of the page until it is written.

Of the two remaining control bits, cache disable (CD) and write through (WT), one is reflected on the output pin for a page table bit (PTB), dependent on the setting of the page table bit mode (PBM) in EPSR. The WT bit, CD bit, and KEN# cache enable pin are internally NORed to determine "cachability." If either of these bits is set to one, the processor will not cache that page of data. For systems that use a second-level cache, these bits can be used to manage a second-level coherent cache, with no shared data cached on chip. In addition to controlling cachability with software, the KEN# hardware signal can be used to disable cache reads.

Floating-point unit

Floating-point unit instructions, as listed in Table 1, support both single-precision real and double-precision real data. Both types follow the ANSI/IEEE 754 standard.[1] The i860 CPU hardware implements all four modes of IEEE rounding. The special values infinity, NaN (not a number), indefinite, and denormal generate a trap when encountered; and the trap handler produces an IEEE-standard result. The double-precision real data occupies two adjacent floating-point registers with bits 31 . . . 0 stored in an even-numbered register and bits 63 . . . 32 stored in the adjacent, higher odd-numbered register.

The floating-point unit includes three-stage-pipelined add and multiply units. For single-precision data each unit can produce one result per clock cycle for a peak rate of 80 Mflops at a 40-MHz clock speed. For double-precision data, the multiplier can produce a result every other cycle. The adder produces a result every cycle, for a peak rate of 60 million floating-point operations per second. The double-precision peak number is 40 Mflops if an algorithm has an even distribution of multiplies and adds. Reducing the double-precision multiply rate saves half of the multiplier tree and is consistent with the data bandwidth available for double-precision operations.

To save silicon area, we did not include a floating-point divide unit. Instead, software performs floating-point divide and square-root operations. Newton-Raphson algorithms use an 8-bit seed provided by a

```
        Do 10, I = 1, 100
10      X = X * A + C

        FMUL X, A, temp
        FADD temp, C, X

        1 result per 6 clock cycles
(a)

        Do 10, I = 1, 100
10      X[I] = A[I] * B[I] + C

        M12TPM A[I], B[I], X[I - 6]

        1 result per clock cycle
(b)
```

Figure 6. Floating-point execution models: data-dependent code in scalar mode (a) and vector code in pipeline mode (b).

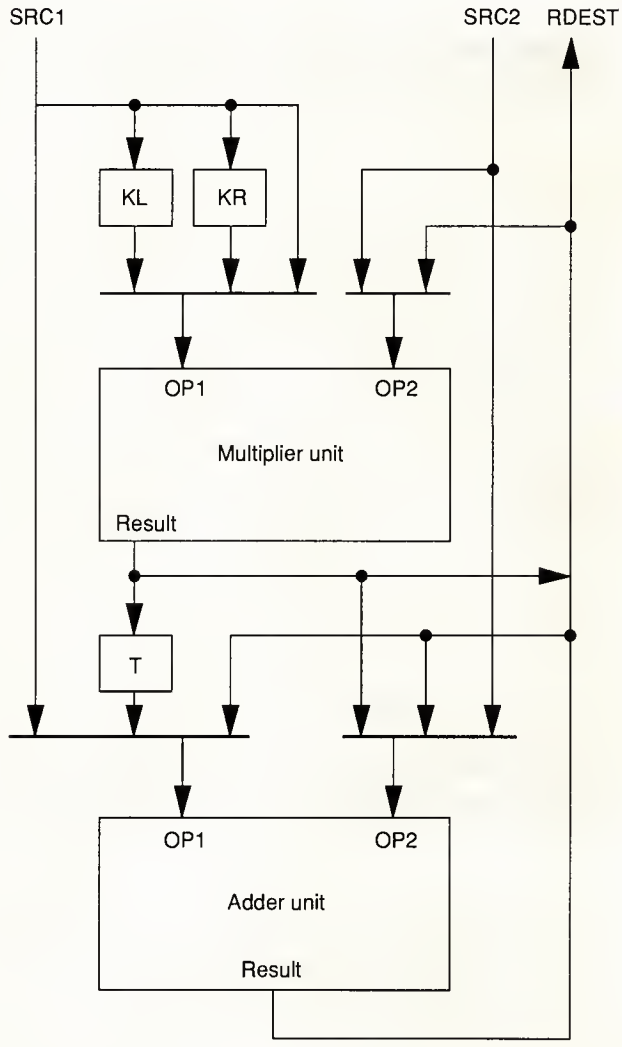


Figure 7. Dual-operation data paths.

hardware lookup table. Full IEEE rounding can be implemented by using an instruction that returns the low-order bits of a floating-point multiply. Therefore these algorithms can take advantage of the pipeline and allow 16-bit reciprocals used in many graphics calculations to be performed either in 10 clock cycles or four pipelined cycles.

The floating-point instruction set supports two computation models, scalar and pipelined. In scalar mode new floating-point instructions do not start processing until the previous floating-point instruction completes. This mode is used when a data dependency exists between the operations or when a compiler ignores pipeline scheduling. In the scalar-mode example of Figure 6 each iteration of the Do loop requires the results from the previous iteration and 6-cycle execution.

In pipelined mode the same operation can produce a result every clock cycle, and the CPU pipeline stages are exposed to software. The software issues a new floating-point operation to the first stage of the pipeline and gets back the result of the last stage of the pipeline. Destination registers are not specified when the operation begins, rather when the result is available. This explicit pipelining avoids tying up valuable floating-point registers for results, so the registers can still be used in the pipeline. Implicit pipelining, using scoreboarding, would cause the registers to become the bottleneck in the floating-point unit.

Pipelining also takes place in a dual-operation mode in which an add and a multiply process in parallel. Figure 7 shows the adder unit, the multiplier unit, the special registers, and the dual-operation data paths. Dual-operation instructions require six operands. The register file provides three of the operands, and the special registers and the interunit bypasses provide the remaining three. The instruction encodings specify the source and destination paths for the units.

Referring back to the pipeline-mode example of Figure 6, note that we show the dual-operation instruction M12TPM SRC1, SRC2, RDEST as M12TPM A[i], B[i], X[-6]. (The M12TPM mnemonic is a variation of the PFAN instruction.) This instruction specifies that the multiply is initiated with SRC1 and SRC2 as the operands. It also specifies that the add is initiated with the result from the multiply and the T register as the operands, and RDEST stores the result from the add. Because of the three stages of the add and multiply pipelines, the available result comes from the operation that started six clock cycles previously.

There are 32 variations of dual-operation instructions. Applications such as fast Fourier transforms, graphics transforms, and matrix operations can be implemented efficiently with these instructions. Some apparently scalar operations, such as adding a series of numbers, can also take advantage of the pipelining capability.

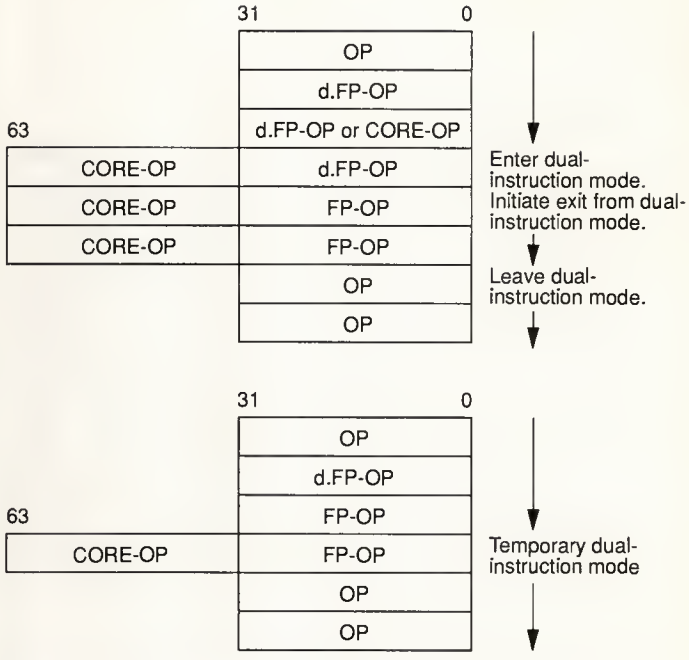


Figure 8. Dual-instruction-mode transitions.

The i860 microprocessor can provide its fast floating-point hardware with the necessary data bandwidth to achieve peak performance for the inner loops of common routines. The dual-instruction mode allows the processor to perform up to 128-bit data loads and stores at the same time it executes a multiply and an add. Figure 8 shows the dual-instruction-mode transitions for an extended sequence of instruction pairs and for a single instruction pair. Programs specify dual-instruction mode in two ways. They can either include in the mnemonic of a floating-point instruction a "d." prefix or use the assembler directives .dual . . . enddual. Either of these methods causes the dual or D-bit of the floating-point instruction to be set. If the processor while executing in single-instruction mode encounters a floating-point instruction with the D-bit set, it executes one more 32-bit instruction before beginning dual-instruction execution. In dual-instruction mode, a floating-point instruction could encounter a clear D-bit. The processor would then execute one more instruction pair before returning to single-instruction mode.

The floating-point hardware also performs integer multiplies and long integer adds or subtracts. Integer multiplies by constants can be performed in the RISC core using shift instructions. To perform a full integer multiply, the processor transfers two integer registers by using IXFR instructions. The FMLOW instruction performs the actual multiplication, and the FXFR instruction transfers the results back to the core. The total operation takes from four to nine clock cycles, depending on what other instructions can be overlapped.

Graphics

The floating-point hardware of the CPU efficiently performs the transformation calculations and advanced lighting calculations required for 3D graphics. The processor performs 500K transforms/second for 4 × 4 3D matrices, including the trivial reject clipping and perspective calculations. A 3D image display requires the use of integer operations for shading and hidden-surface removal. The graphics unit hardware speeds these back-end rendering operations and operates directly into screen buffer memory. It uses the floating-point registers and operates in parallel with the core.

Graphics instructions take advantage of the 64-bit data paths and can operate on multiple pixels simultaneously, realizing 10 times the speed of the RISC core when performing shading. Instructions support 8-, 16-, and 24/32-bit pixels, operating respectively on eight, four, or two pixels simultaneously.

In 3D graphics, polygons generally represent the set of points on the surface of a solid object. During transformation, the graphics unit calculates only the vertices of the polygons. The unit knows the locations and color intensities of the vertices of the polygons, but points between these vertices must be calculated. These points, along with their associated data, are called pixels. If a figure is displayed with only the vertices and simple lines, it appears as a wireframe drawing. The simplest wireframe drawing typically shows all vertices, even the ones that should be hidden from view by an overlapping polygon. To show shaded 3D images, the graphics unit must display the surface of the polygons. Where polygons overlap, it must display the polygon closest to the viewer.

In graphics calculations the z value represents the distance of a pixel from the viewer. Although the depth of each polygon's vertices is known, to overlay polygons not on a vertex, the graphics unit must interpolate the depths from the bordering vertices. This step is called z interpolation. In this step the depths of all points of a polygon can be determined. For overlapping points, the z values of different polygons can be checked and only the pixel data of the polygon closest to the viewer displayed.

To perform the procedure just described, the graphics instructions include intensity interpolation, z interpolation, and z-buffer checks. Intensity interpolation allows smooth linear changes in pixel intensity and color between vertices. This capability provides a smoother appearance than does the flat shading of the polygons. The more data bits per pixel, the smoother the interpolation becomes. The i860 CPU graphics instructions support both Gouraud and higher order shading techniques. Gouraud shading interpolates intensities along the scan lines. Figure 9 illustrates pixel interpolation for Gouraud shading of a triangle. The intensity level across the scan line shown is interpolated from 30 to 27.

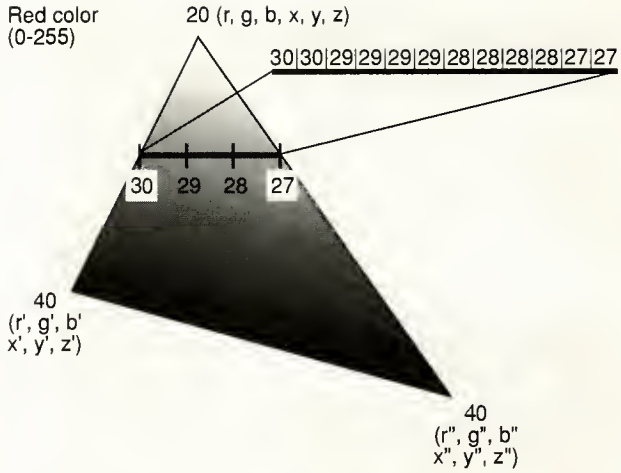


Figure 9. Pixel interpolation for Gouraud shading of a triangle for red colors and 0-255 intensity levels.

In graphics the z-buffer, which can reside in normal dynamic RAM, stores the depth of the pixel buffer currently being displayed. Instructions for z-buffer interpolation calculate the z values between vertices. Z-buffer check instructions compare the new pixels' z values to the values in the z-buffer, and if closer, the pixels are unmasked in the pixel mask register. The RISC core operates in parallel with the graphics unit and executes a pixel store instruction. The pixel store updates the pixels that are unmasked in the mask register. If a pixel is updated, the new z value needs to be stored to the z-buffer. The z-buffer check instruction updates the buffer with the minimum z value for each pixel.

Most workstations typically have a base graphics system of a simple frame buffer with simple display hardware. With a frame-buffer graphics system, the i860 CPU can perform Gouraud-shading operations on 50,000 triangles per second at 40 MHz. This level of performance exceeds that of workstations that include costly dedicated graphics processor boards.

Caches

The i860 CPU has a 4-Kbyte instruction cache and an 8-Kbyte data cache, each with its own address and data paths to support concurrent accesses. The data cache supports up to 128-bit accesses on each clock cycle, and the instruction cache supports up to 64-bit accesses. The aggregate bandwidth at 40 MHz is 960 Mbytes/second. Both caches combine two-way set-associative parallelism with a 32-byte line size. Additionally, the data cache uses write-back caching.

Both caches use virtual addresses to avoid a critical path in the cache access. Data cache accesses use the TLB lookup for enforcing the page-based protection. Since both caches use virtual tags, software must avoid the aliasing of data. Within a context, each physical address must only be accessed with one virtual address. During context switches, the instruction cache must be invalidated and the data cache flushed. The caches, although large enough to give hit rates above 90 percent within many applications, are too small to provide hits across context changes. Therefore, we did not feel process IDs or a duplicate set of physical tags to avoid flushing the cache between context switches were warranted.

Flushing the data cache is an easy way to avoid aliasing, and a simple calculation shows what little impact a small cache has on performance flushing. A typical i860 CPU context switch, including the data cache flush, takes approximately 65 microseconds. In the worst case, a workstation will change context 200 times per second; multiplying ($65 * 10^{-6}$ seconds * 200 times/second) equals a 1.3 percent performance degradation due to context switching.

Write-back data caching avoids propagating all writes to the external bus, which reduces bus traffic. It also prevents a bottleneck in vector operations where write traffic from the vector result collides with an incoming vector operand. With write-back caching, the hardware necessary to implement transparent caching for multiprocessor systems moved costs beyond the silicon budget of this implementation. Instead, we use software to manage cache coherency. Each processor can cache code, vector register data, and private stack data, while shared data remains uncached. Software controls the caching by using a cachable bit in the page table entries to prevent shared data from being cached. External hardware can also assert a cachable enable pin to control cachability of each line's read miss. The flush instruction forces all "dirty" blocks in the data cache back to memory. Flushing is needed before removing a page or changing to a new virtual address space.

We included optimizations for cache-miss processing. Each cachable read miss results in four bus cycles to fill the 32-byte cache line. First, the processor fetches the referenced data word and performs a wraparound fill to read the entire line. The processor can then continue execution when the first word is returned. The processor contains two 128-bit write buffers used for store misses and cache miss processing. When the processor issues a store instruction that misses the cache, it can continue execution while the write buffer carries out the actual memory write. The write buffers support two store misses and also support a delayed write back of the dirty cache line. If a cachable read miss displaces a dirty cache line, three operations take place. The processor writes the dirty line to the write buffer, the cache line read takes place on the external bus, and then the write back occurs.

A convenient software model for managing the data cache for vector computations on large matrices is to the treat the data cache as a "vector register set." Vectors, or their intermediate results, that are being reused are kept in the onboard cache by referencing with the normal floating-point load instruction. The vectorization process analyzes nested loops to determine which vectors are reusable in the second-loop level. Vector register references in the vector library routines use the normal floating-point load instruction. Vector memory references use the pipelined floating load instruction to stream the data from memory directly into the registers and not disturb the cache. Using the data cache as a vector register set is a more flexible concept than that found in many supercomputers with small, fixed-length vector registers. This concept offers the advantages of a vector register set for vector computations while retaining the flexibility of a data cache for scalar computations.

Bus interface

Designed for scalability to 50 MHz, the i860 CPU external bus performs a 64-bit transfer every two clock cycles. Thus, we achieve the design of a practical TTL (transistor-transistor logic) system, even at 50 MHz. The bus can interface either to a second-level cache or directly to a DRAM system. The bus allows optional pipelining for increasing the access time without decreasing the bandwidth. The full bus bandwidth can be realized from one bank of DRAMs, however, the latency will be greater than if a fast static RAM cache is used.

With the two-cycle transfer rate, the external bus can supply one memory operand for every double-precision add/multiply pair, or two contiguous single-precision operands for every two single-precision add/multiply pairs. The other two vector operands for an add/multiply pair must come from the onboard data cache. This approach provides the same ratio of floating-point rate to external memory bandwidth as the Cray 1. To avoid bus bottlenecks, the vectorization process must try to reuse two of the three vector operands in the second-level inner loop.

The i860 microprocessor contains a synchronous interface with a demultiplexed address and 64-bit-wide data bus. The address bus provides 32-bit addressing, consisting of 29 address lines and separate byte enable signals for each eight data bits. The bidirectional data bus can accept or drive new data on every other clock cycle, yielding a bandwidth of 160 Mbytes per second at 40 MHz.

The bus optionally allows for two levels of bus pipelining selected on a bus cycle-by-cycle basis. When pipelining, a new cycle starts prior to the completion of the outstanding cycles. Two levels of pipelining allow

Table 2.
Processor-pin summary.

Pin name	Function	Active state	Input/ output
Execution control pins			
CLK	Clock	—	I
RESET	System reset	High	I
HOLD	Bus hold	High	I
HOLDA	Bus hold acknowledge	High	O
BREQ	Bus request	High	O
INT/CS8	Interrupt, code size	High	I
Bus interface pins			
A31-A3	Address bus	High	O
BE7#-BE0#	Byte enable	Low	O
D63-D0	Data bus	High	I/O
LOCK#	Bus lock	Low	O
W/R#	Write/read bus cycle	High/Low	O
NENE#	Next near	Low	O
NA#	Next address request	Low	I
READY#	Transfer acknowledge	Low	I
ADS#	Address status	Low	O
Cache interface pins			
KEN#	Cache enable	Low	I
PTB	Page table bit	High	O
Testability pins			
SHI	Boundary scan shift input	High	I
BSCN	Boundary scan enable	High	I
SCAN	Shift scan path	High	I
Intel-reserved configuration pins			
CC1-CC0	Configuration	High	I
Power and ground pins			
V_{CC}	System power	—	—
V_{SS}	System ground	—	—

A # symbol after a pin name indicates that the signal is active when at the low-voltage level.

three cycles to operate at one time. Fast TTL latches can be used on the address and data bus. This method isolates the memory array from the processor pin timings, allowing easy scalability and providing the maximum time for memory accesses. With pipelining, the maximum data rate of the bus can be sustained even if the access time is six clock cycles. We achieve over 100 nanoseconds of address-to-data access time for a full bandwidth system at 40 MHz.

A summary of the processor pins appears in Table 2. We timed the processor with a single-frequency, TTL-level clock. An optional mode for executing out of one 8-bit-wide EPROM can be entered at reset by activating the INT/CS8 pin. In this mode the processor fetches instructions from the EPROM with the byte-enable signals BE2#-BE0# redefined as address lines A2-A0.

The HOLD, HOLDA, and BREQ signals activate arbitration of the processor's local bus. When a DMA controller, or another processor, needs access to the local bus of the CPU, it asserts HOLD. When the CPU completes all of its outstanding bus cycles, it floats the bus interface pins and returns HOLDA active high. The CPU will remain in this state with HOLDA active until HOLD is deasserted. The CPU can continue processing while in HOLD until the external bus is required. At this time it asserts the BREQ output signal. Arbitration logic samples the BREQ signal to arbitrate a shared bus.

The A31-A3 and BE7#-BE0# bus interface pins can access up to 4 gigabytes of address space. The address lines select the 8-byte word, and the byte-enable signals select the byte within the word. For read accesses to cachable memory, the processor caches the entire data bus so the byte-enable signals are ignored. For write operations the byte-enable signals determine which bytes in memory must be updated. The i860 microprocessor does not, however, allow misaligned accesses. Data of 32 and 16 bits must be placed on 4- and 2-byte boundaries, respectively. However, single-byte data can be placed at any byte location. The 64 bidirectional data pins can transfer 8-, 16-, 32-, or 64-bit quantities; pins D7-D0 signify the least significant byte and D63-D56 signify the most significant byte.

The processor asserts the ADS# output during the first clock cycle of each bus cycle to indicate the start of the bus cycle. The W/R# signal distinguishes the write and read bus cycles. The NENE# output indicates to the DRAM controller that the current address is in the same DRAM page as the previous cycle. As shown later, this information is useful for designing high-performance memory systems.

The NA# input to the CPU controls pipelining and can be asserted before the current cycle ends. When the processor samples NA# active, it can start driving the next bus cycle's address and definition. This can be done two times prior to returning data for any of the cycles.

While NA# controls the address and bus cycle definition signals, READY# controls the data operations. When READY# is sampled as active for a read, the processor latches the data from the data bus. When READY# is sampled as active for a write, the processor stops driving the data from that cycle. READY# also serves to end a bus cycle. The LOCK# signal output provides atomic (indivisible) sequences. Using LOCK# prevents the processor from relinquishing the bus even if HOLD is asserted. For multiprocessor systems, the external hardware only needs to lock the first address in a locked sequence.

This processor samples the KEN# input to determine

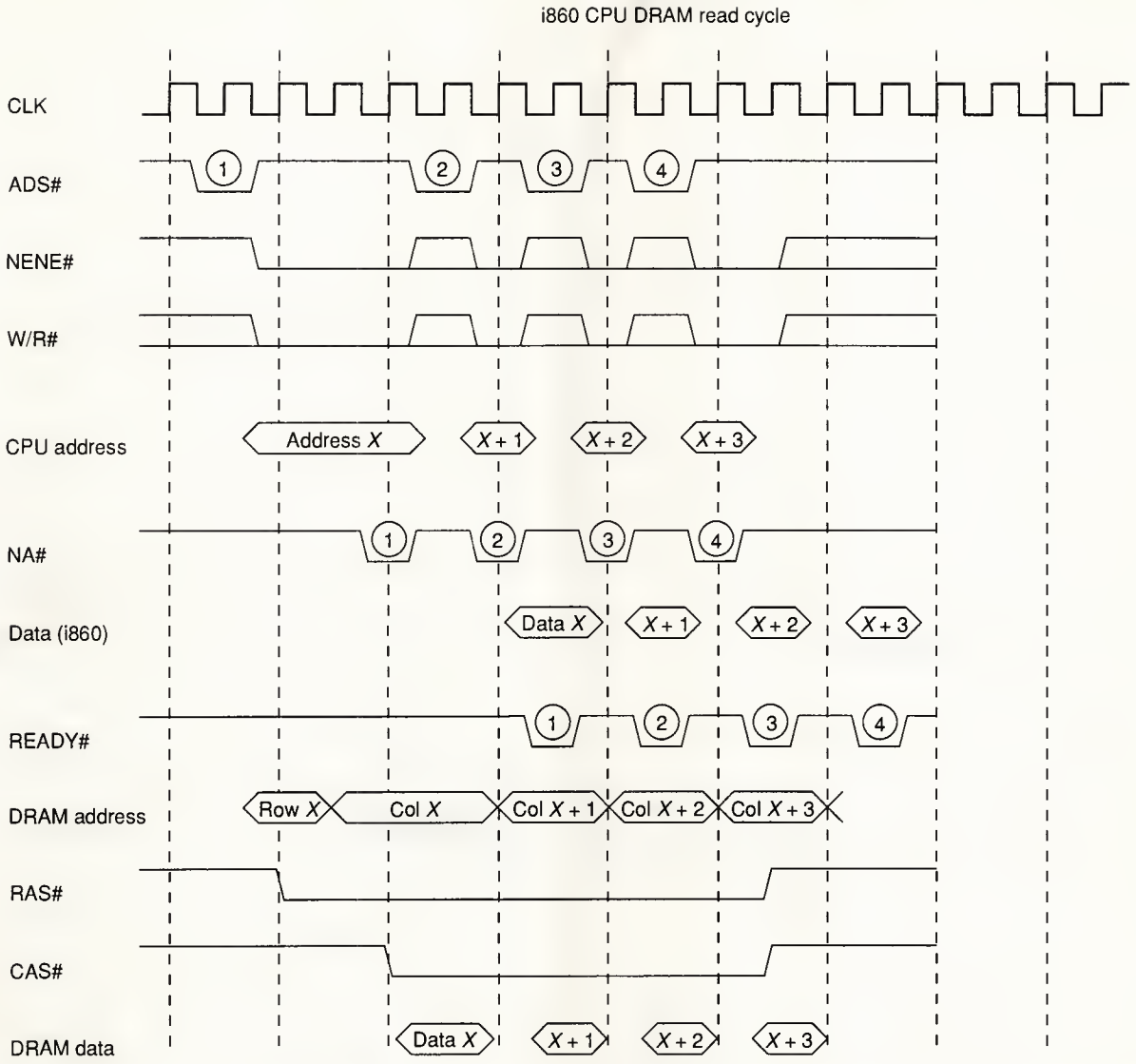


Figure 10. The CPU performs four read cycles to fill a cache line.

if the data for the current read cycle is cachable. Address space that is used for input and output can be decoded to deassert KEN# during I/O accesses. Software can also mark areas of memory as noncachable on a page-by-page basis. If the software has not disabled caching of the page, and KEN# is available for a read cycle, three additional 64-bit bus cycles will be generated to fill the 32-byte cache block.

Interfacing to a DRAM system

Figure 10 shows the processor performing four read cycles as it would do to fill a cache line. Also shown in the figure is the NA# signal returned to the processor, which indicates that the system can accept the next bus cycle. Two NA#s are returned before any of the cycles are completed. To complete a read cycle, the memory system provides the data on the bus and returns READY# to the processor. Once fully pipelined, the memory system provides data and READY# on every other clock cycle. Important for high performance, this data rate can be provided by ordinary static column DRAMs. The processor also provides the control signal NENE# to optimize DRAM control.

The memory system in Figure 11 on the next page consists of an address buffer; an address latch; eight latching data buffers; and a 64-bit-wide, static column-mode DRAM (256K × 4). This arrangement allows the memory size to be increased in increments of two megabytes. Using 256 × 4-memories also has advantages in reducing power and signal-drive requirements. To support the two levels of pipelining, the processor latches both address and data. The address latches hold

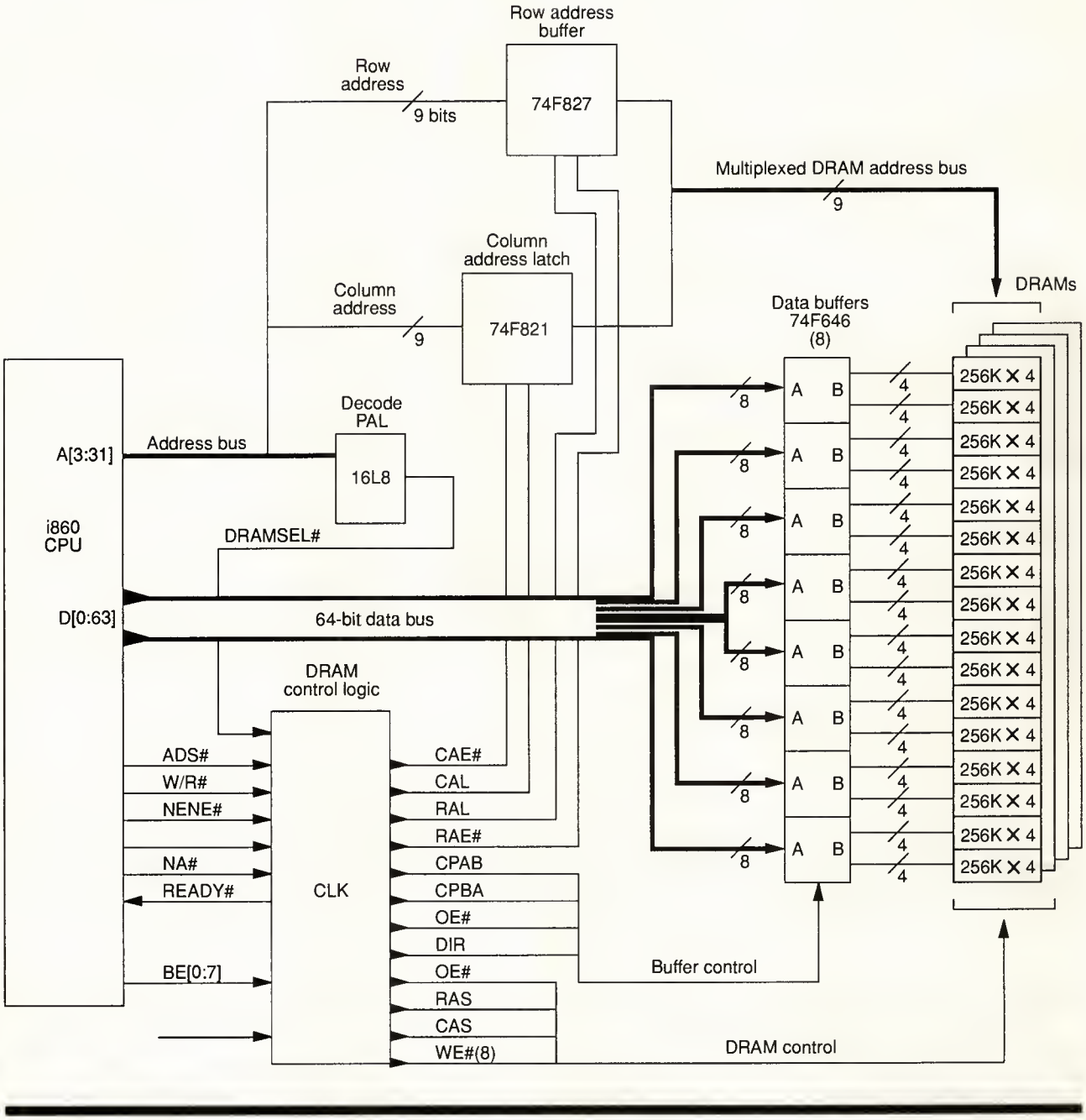


Figure 11. A DRAM system for the i860 microprocessor requires little "glue logic."

the address of the previous cycle, while the data from the cycle prior to that is held in the data buffers. Using TTL components on the address and data paths also has the advantage of isolating the memory system from the processor's pin timings.

The two address latches are used for multiplexing the row and column addresses from the processor to the DRAMs' address lines. When accesses occur within the DRAM page, only the column address needs to be supplied to the memory address lines. Most systems that use a fast-access DRAM mode need an additional hardware comparator. The i860 CPU has a comparator—which supplies the NENE# signal on each bus cycle—built into the bus unit. The controller uses this signal to determine if a fast static column-mode access can occur or if a full DRAM cycle needs to take place.

The bidirectional data buffers latch the data for both reads and writes. For reads, the buffers latch data and

return READY# on the following clock cycle. With the two levels of pipelining the total access time is six cycles, while data is available every two cycles. Zero-wait-state operation does not require pipelining for write cycles. When a write occurs, the address and data latched in the buffers allow READY# to be returned to the processor. The actual write cycle occurs after READY# returns to the processor. This delayed write operation allows processor execution to continue even though the write has not fully completed.

Using 85-ns static column-mode DRAMs, the 33-MHz i860 microprocessor can operate at zero wait states for access within the DRAM page. The two-level pipelining and two-clock transfer rate allow the processor to sustain performance without the need for an external cache memory system.

Software support

Both internal development teams and independent vendors provide a full complement of software development tools and operating systems for the i860. Figure 12 shows the software development tools available: C and Fortran compilers, assembler/linker, simulator/debugger, Fortran vectorizer, plus mathematical, vector primitive, and 3D graphics libraries. To support the initial development environments, both Unix System V run on a 386 microprocessor and OS/2 host cross-compilers. The optimizations used in the compilers include coloring for register allocation, register-based parameter passing for calls, interblock common subexpression and loop invariant elimination, constant propagation, strength reduction, extensive peephole optimizations, and instruction scheduling.

Scientific-application support includes a Fortran vectorizing precompiler. Vectorization occurs in Do and If loops, outer loops, and forward-branching conditional operations. The precompiler recognizes these structures and generates calls to a set of preprogrammed procedures. The preprogrammed procedures are optimized for the processor's instruction set and for managing the data cache as a vector register. Additionally, other high-level languages can call these procedures. We plan to further increase the degree of parallelism that high-level languages can use in the processor. We also provide a library of assembly-language routines for scalar mathematics.

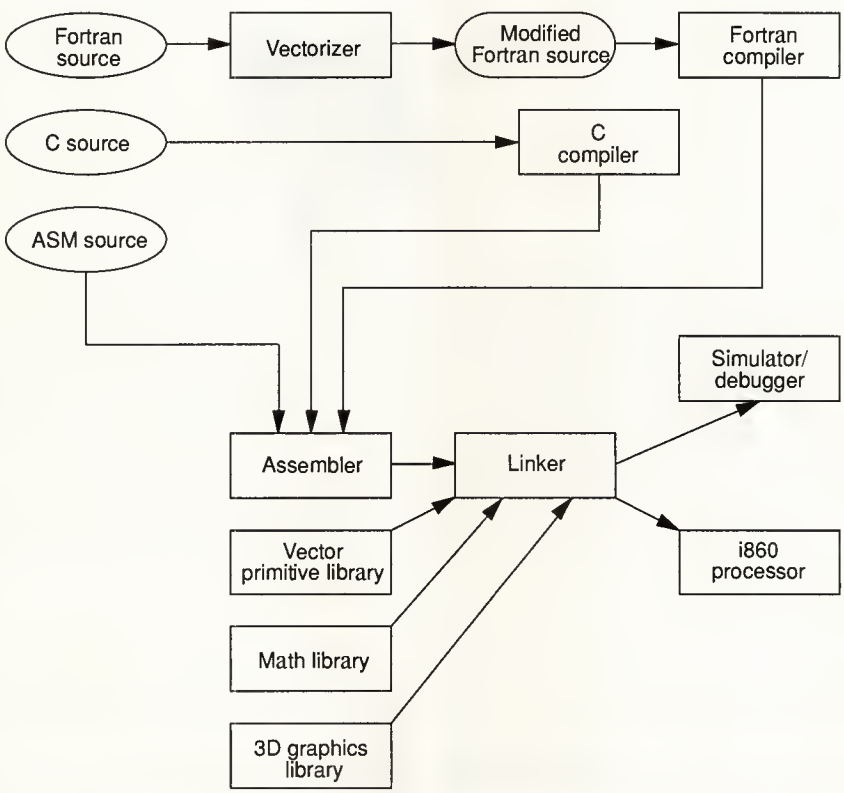


Figure 12. Software development environment supporting the i860.

The first 3D visualization tool ported to the i860 CPU is Ardent Computer's Dore. This tool supports both real-time, interactive 3D modeling and higher quality static images. Several windowing environments and other 3D tools and libraries are also being ported.

Application software can be run on either a software simulator or an add-in application accelerator. Both share a common debugging interface. The simulator allows the user to model different memory systems and measure their effects on performance. A Unix V/386 or OS/2 hosts the application accelerator, which includes a runtime operating environment that maps I/O requests back to the host processor.

A multiprocessing version of Unix System V Release 4.0 is under development for the i860 CPU. This is a joint effort by AT&T, Convergent Technologies, Intel, Olivetti, Prime Computer, and others. We plan to maintain source-code compatibility with the high-level languages between the 386, i486, and i860 microprocessors. Specifications for an applications binary interface standard (ABI) will allow portability of application software across multiple vendors' system implementations.

The i860 microprocessor begins the second generation of 32-bit RISC processors. By using a 64-bit architecture, the i860 delivers balanced MIPS, Mflops, and graphics performance. The million-transistor budget lets us integrate the RISC core and provide dedicated, fast floating-point hardware, graphics capabilities, and cache memories on one chip. The design allows maximum parallelism between the functional units while achieving a balance between computation speed and data bandwidth. Mainframe and supercomputer architectural concepts let the processor offer a complete solution to the requirements of high-computation applications.

References

1. *ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic,* IEEE Computer Society Press, Los Alamitos, Calif., 1985.

Les Kohn is a chief architect for high-performance processors at Intel Corporation of Santa Clara, California, where he has worked on various 32- and 64-bit microprocessor design projects. Before joining the company, he worked as a software manager and architect for the NS32000 family at National Semiconductor. His interests include computer architectures and compilers and electronic synthesizers.

Kohn received his BS degree in physics from the California Institute of Technology in Pasadena.

Neal Margulis is a senior engineer for high-performance processors at Intel. His interests include processor architecture and system design. Margulis received his degree in electrical engineering from the University of Vermont in Burlington. He is a member of the IEEE Computer Society and Tau Beta Pi.

Questions concerning this article may be directed to the authors through Michael Sullivan at Intel Corporation, SC4-42, 2625 Walsh Avenue, Santa Clara, CA 95051.

The MC68332 Microcontroller

This new modular microcontroller combines a high-performance CPU with a specialized time-processing unit for improved real-time control.

Joe Jelemensky
Vernon Goler
Brad Burgess
James Eifert
Gary Miller

Motorola

The MC68332 is the first in a new family of microcontrollers from Motorola. A study of the 16/32-bit-microcontroller market greatly influenced the 68332 architecture. Companies in the field of real-time control helped identify the needs for several major features. Among these specifications were high-performance computational capability, a large address space, and the ability to process large amounts of complex, high-speed I/O. Users saw these features as necessary to solve the more complex control algorithms attendant upon future developments. We established key design goals from the outset of the project to meet these requirements.

Total-system goals

Our primary design objective was to increase the performance of the total system. The most obvious way to do this is to increase the performance of a microcontroller's CPU. However, the CPU is only one component of a microcontroller, which also contains highly complex on-board peripherals. A large increase in microcontroller performance therefore requires a total-system approach.

In general terms, a fast, powerful CPU allows routine tasks—such as servicing high-speed I/O and serial communication channels—to execute more quickly. This process alone, however, does not greatly increase total performance. The latency for servicing peripheral devices in an interrupt-driven, real-time system remains a problem. As control applications become more complex, the sheer number of events requiring service increases greatly. Together, these two factors can quickly erode any gains in CPU performance.

Peripheral devices and routine events compete with each other for service time from the CPU. Adding intelligence to system peripherals so they can process simple events with their own resources eliminates much of the event servicing normally performed by the CPU. This approach also significantly reduces the total overall latency in the system and conserves CPU resources for solving its control algorithm. System performance increases considerably.

CPU design goals

As a second major objective, we concentrated on defining a high-performance CPU (see CPU module section for specifications). This defi-

0272-1732/89/0800-0031$01.00 © 1989 IEEE

nition includes an ample general-purpose register set, a rich complement of instructions, the capability to operate on large data sizes, a fast clock speed, and instruction pipelining to provide high-performance computational capabilities. The CPU also should execute compiler-generated programs efficiently and have a generous addressing range to accommodate them.

As control applications become more complex—like the space shuttle and robot controllers—program size increases. High-level languages are necessary to rapidly generate large, reliable programs. Consequently, we designed the CPU for HLLs like C or Modula 2.

The modular approach to design

Our third major objective was to design a microcontroller family that could be easily adapted to individual applications. This goal required a flexible microcontroller design that formalized interconnections and reduced logic interdependencies.

To ensure a short design cycle—while providing this versatility—we employed a modular approach. We designed several functional modules simultaneously and independently. This approach has many benefits—particularly for projects that have the scope of the 68332. Dividing the chip into functional modules provided a specific focus for each group of designers, each with its own module. A standard intermodule bus (IMB) interface also freed the designers from having to know unnecessary details about other modules.

Large designs monopolize a design center's resources in terms of the engineering staff. Modularity eases this problem by drawing resources from several centers. The 68332 was actually designed in Texas, California, and Israel, with overall responsibility lodged in Austin, Texas. This method also capitalized on the expertise of each center.

In a self-contained module, the only avenue of stimulus occurs through the IMB and dedicated pins. Therefore, the production test vectors for a module need not change when it is used on a new device. This fact reduces the generation of vectors to those required for system-wide testing or any new modules in 68300-family design. It also greatly reduces the design-to-production cycle time of a new device and standardizes testing.

Figure 1 shows a general modular layout. As mentioned, each module is self contained. Each module (discussed later) interfaces to the IMB for CPU access. I/O pin connections occur outside of each module that requires external I/O. The IMB is a synchronous, multimaster, two-clock-cycle bus. It contains 24 address and 16 data lines along with associated control signals for data-transfer handshaking, interrupt, and bus-mastership arbitration. The external bus interface (EBI) contained in the system integration module (SIM) performs the interface between the IMB and the external bus.

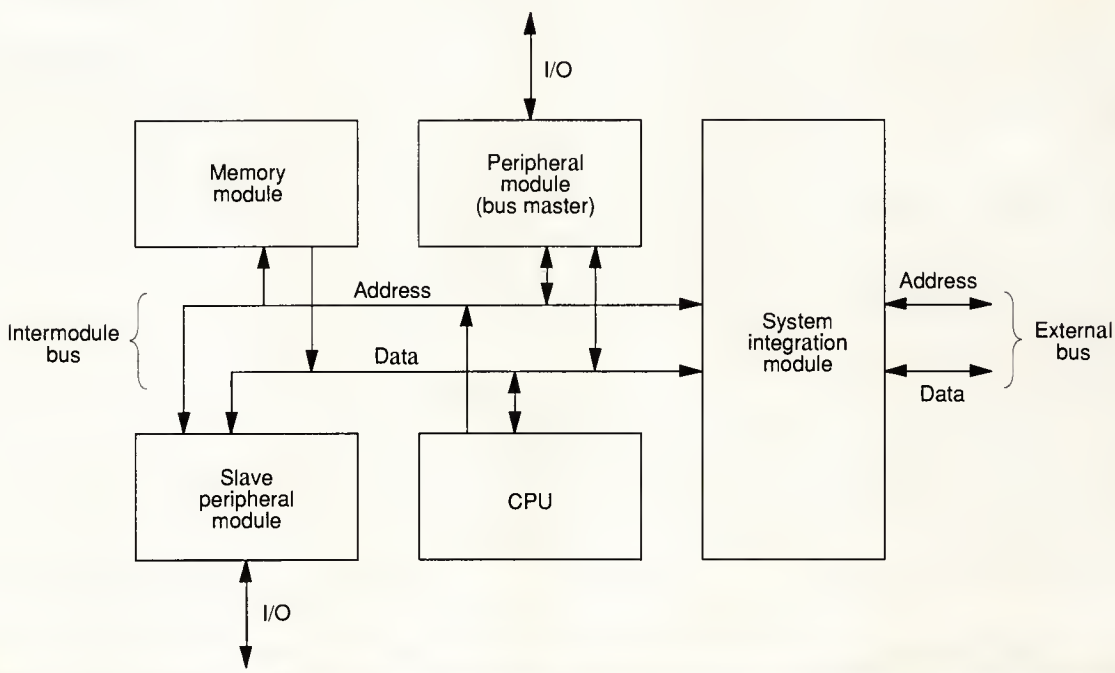


Figure 1. Typical modular layout with multiple bus masters.

Overview of the 68332

The 68332 has a fully static, 1-micrometer HCMOS design and offers low power consumption at an operating voltage of 5 VDC (±10 percent). It operates within a temperature spectrum of −40 to +125° C, with a frequency range of 0.1 to 16.77 megahertz.

Figure 2 is a die photograph of the modular layout that more specifically outlines the functional modules of the 68332. The modules include

- a SIM that straddles the IMB and contains programmable chip selects, a system clock, a periodic interrupt function, and system protection features;
- a queued serial module (QSM) that contains both asynchronous and high-speed synchronous, serial submodules;
- a 1 Kbyte × 16-bit static RAM that has standby capability;
- an M68000 family CPU; and
- a time processor unit (TPU) that processes time-based, high-frequency I/O functions.

Here we briefly summarize the features of the SIM, QSM, and RAM before moving on to more detailed discussions of the CPU and time processor modules.

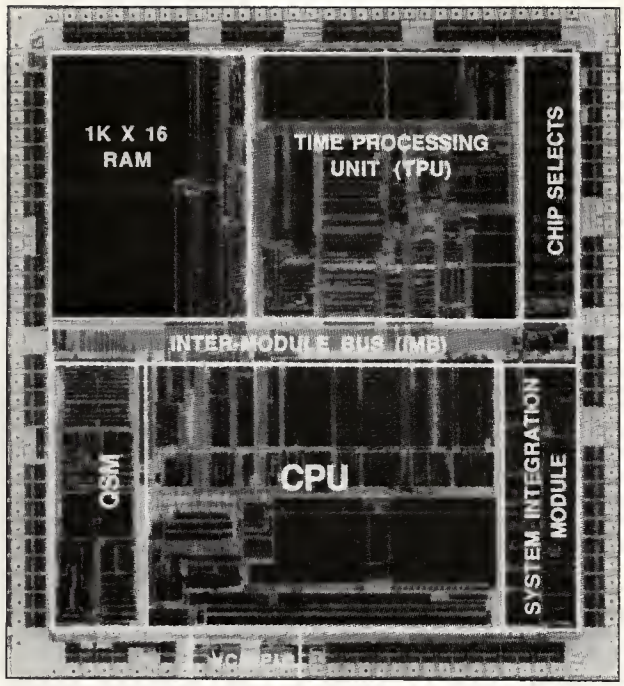


Figure 2. Photomicrograph of the MC68332.

System integration module

The 68332 external bus shown in Figure 1 is similar to that of the 68020. It supports seven levels of interrupts with arbitration between internal and external interrupts. The EBI supports multimaster arbitration on the external bus. Twelve user-programmable, chip-select pins decode address ranges and trigger on selected bus cycles. One chip-select pin can be configured during the reset sequence to select a boot ROM for initial code execution. Chip-select logic can eliminate the need for external address decoding logic and data-transfer-and-size-acknowledge (DSACK) circuitry. The chip-select pins can be programmed to run on the fast two-clock-cycle bus—similar to the 68030's synchronous termination bus cycle—or a three- to 15-clock-cycle bus. They can also allow external generation of the DSACK signal. This wide range of bus speeds allows the selection of very fast or very slow memories and peripherals.

A bus monitor generates a bus-error exception signal when a memory access fails to complete. A halt monitor resets the system when the CPU halts. A software watchdog resets the system if code execution fails to perform a specified sequence of events within a preprogrammed period of time.

Queued serial module

The QSM contains two serial subsystems. The first is an asynchronous serial communications interface (SCI). It is a Universal Asynchronous Receiver/Transmitter-type device similar to the SCI subsystem found in the MC6805 and MC68HC11 families. It contains its own baud-rate generator and can operate at speeds up to 524,000 baud when the 68332 runs at 16.77 MHz. Added features include parity and enhanced idle-line monitoring for use in multimaster networks.

The second serial subsystem is a queued serial peripheral interface (QSPI). It relieves the CPU from servicing devices connected to the 68332 via the serial peripheral interface (SPI) expansion bus. The SPI is a simple, three-wire, master/slave synchronous bus that connects devices in close physical proximity within a board or single chassis. These devices include A/D converters (ADCs), display drivers, discrete I/O expanders (MC14489 and MC145050), or even other microcontrollers. The configuration of the QSPI-shared RAM is a pair of 16-bit I/O data queues. Associated with each of the 16 entries in the queue pair is an 8-bit control field. This field contains information about the SPI device associated with the queue entry. The QSPI uses the information in this control field to automatically select the proper serial device by controlling the state of four peripheral select lines. The QSPI also provides the correct timing and bit-sequence lengths to the peripheral. The CPU merely provides the control information and the output data to peripherals. The QSPI actually selects devices and transfers data between the 68332 and the device. Because of its queue structure and a serial access that is transparent to the CPU, the QSPI makes the serial devices appear as though they were memory-mapped devices on the 68332. With its built-in baud-rate generator, the QSPI can transfer data at up to 4.2 MHz, assuming a 16.77-MHz system clock.

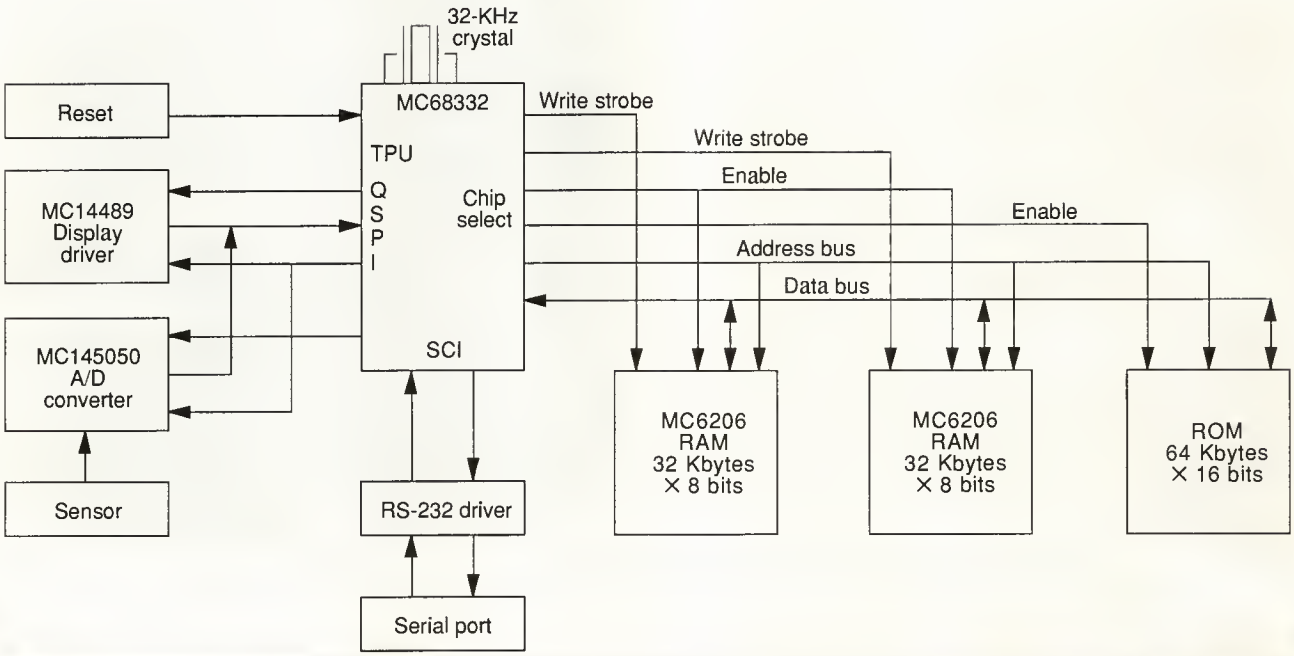


Figure 3. Block diagram of a simple system.

Standby RAM

This module includes 2 Kbytes of static RAM that can be powered from an external source. It provides fast-access (zero-wait-state) memory. Its ideal use is as a small stack or storage area for parameters (variables) that are frequently accessed. Its standby capability retains parameters when the rest of the system powers down.

Small-system design

Figure 3 demonstrates the 68332's system integration by showing a minimum system. The chip-select circuitry removes the need for all external decoding logic. In this example, two chip-select pins function as write strobes, which creates an extra 30 nanoseconds of access time at the cost of continuous RAM selection. Two more chip selects enable RAM and ROM output. Any memories with access times of 45 to 900 ns can be used in this system. The QSPI connects to the MC14489 multicharacter light-emitting diode display/lamp driver and MC145050 8-bit ADCs with serial interface. No external decoding or bus timeout logic is required.

CPU module

The 68332 CPU module is the latest member of the M68000 family and has inherited a number of previous-generation features.

Family features. The CPU module has 32-bit registers, arithmetic units, and data paths. Only 24 bits of the address bus have been connected to pins or brought out of the chip. The CPU operates with the 16-bit data bus of the 68332 microprogram control unit (MCU) in the data portion of the IMB. The CPU's instruction set and level of performance fall between those of the 68010 and the 68020. The CPU module supports all of the 68010 instructions as well as many of the 68020 extensions shown in Table 1.

The 68332 CPU supports all addressing modes of the 68010 and 68020 except for the 68020's memory-indirect mode (see Table 2 on p. 36). For reference, the unsupported 68020 instructions are Bit Field, Coprocessor, Call Module, Return from Module, Compare and Swap, Pack Binary-coded Decimal, and Unpack BCD. The 68332 uses an instruction restart mechanism to support virtual memory.

Added features. The 68332 has two new instructions, Low-Power Stop and Table (LPSTOP and TBL). These instructions can be emulated in software by other members of the M68000 family. LPSTOP causes the CPU to run the LPSTOP broadcast cycle, which directs the system to enter a power-saving mode. This feature is similar to the MC68HC11 STOP instruction, which stops the major system clocks.

Controller applications often replace time-consuming function calculations with lookup tables that represent the function. The Table Lookup and Interpolate instruction, TBL, supports piecewise, linear, com-

Table 1.
M68000 instruction-set extensions.

Mnemonic	Description	68020	68332
Bcc	Supports 32-bit displacements	Yes	Yes
BFxxx	Bit-field instructions	Yes	No
BGND	Background operation	No	Yes
BKPT	New instruction functionality	Yes	Yes
BRA	Supports 32-bit displacements	Yes	Yes
BSR	Supports 32-bit displacements	Yes	Yes
CALLM	New instruction	Yes	No
CAS, CAS2	New instructions	Yes	No
CHK	Supports 32-bit operands	Yes	Yes
CHK2	New instruction	Yes	Yes
CMPI	Supports PC relative addressing	Yes	Yes
CMP2	New instruction	Yes	Yes
cp	Coprocessor instructions	Yes	No
DIVS/DIVU	Supports 32- and 64-bit operations	Yes	Yes
EXTB	Supports 8- to 32-bit extensions	Yes	Yes
LINK	Supports 3-bit displacements	Yes	Yes
LPSTOP	New instruction	No	Yes
MOVEC	Introduced on the 68010, supports new control registers	Yes	Yes
MOVEfromCCR	Introduced on the 68010	Yes	Yes
MOVES	Introduced on the 68010	Yes	Yes
MULS/MULU	Supports 32-bit operands, 64-bit results	Yes	Yes
PACK	New instruction	Yes	No
RTD	Introduced on the 68010	Yes	Yes
RTM	New instruction	Yes	No
TBLU, TBLS	TBL unsigned and signed—new instruction	No	Yes
TST	Supports PC relative, immediate, and An addressing	Yes	Yes
TRAPcc	New instruction	Yes	Yes
UNPK	New instruction	Yes	No

pressed-data tables to model complex functions. TBL requires two operands: (a) a pointer to a data table representing the function and (b) the value to be passed to the function. TBL calculates the resultant data point by linear interpolation. The use of TBL to replace complex function calculations provides a significant data-throughput increase. The box on p. 40 demonstrates the operation of the TBL instruction.

A typical application involves reading a nonlinear sensor using an ADC connected to the QSPI. The instruction uses the sensor calibration data, which is then used in the control algorithm.

Another new feature in the 68332 is the extension of the M68000 family's illegal instruction trapping feature to include the checking of both first-word illegal instructions and subsequent illegal effective-address words. This procedure increases coverage for catching programming errors, memory faults, or runaway code.

Table 2.
M68000 addressing modes.

Mode	Mnemonic	68000/ 68010	68020	68332
Register direct	Rn	Yes	Yes	Yes
Address register indirect	(An)	Yes	Yes	Yes
Address register indirect w/postincrement	(An) +	Yes	Yes	Yes
Address register indirect w/predecrement	– (An)	Yes	Yes	Yes
Address register indirect w/displacement	(d16,An)	Yes	Yes	Yes
Address register indirect w/index (8-bit displacement)	(d8,An,Xn)	Yes	Yes	Yes
Address register indirect w/index (base displacement)	(bd,An,Xn * SCALE)	No	Yes	Yes
Memory indirect w/postincrement	([bd,An],Xn,od)	No	Yes	No
Memory indirect w/predecrement	([bd,An,Xn],od)	No	Yes	No
Absolute short	(xxx).W	Yes	Yes	Yes
Absolute long	(xxx).L	Yes	Yes	Yes
PC indirect w/displacement	(d16,PC)	Yes	Yes	Yes
PC indirect w/index (8-bit displacement)	(d8,PC,Xn)	Yes	Yes	Yes
PC indirect w/index (base displacement)	(bd,PC, Xn * SCALE)	No	Yes	Yes
Immediate	#(data)	Yes	Yes	Yes
PC memory indirect w/post-increment	([bd,PC],Xn,od)	No	Yes	No
PC memory indirect w/pre-decrement	([bd,PC,Xn],od)	No	Yes	No

Block diagram. Figure 4 outlines the four main blocks of the processor:

- the execution unit,
- a microcoded controller called the microengine,
- the execution-unit control, and
- the pipeline control/bus-interface unit.

The execution unit block contains separate instruction-execution and bus-execution units. The first unit executes all instruction and effective-address calculations. The second increments the program counter and generates the address for the second word of a long operand. The instruction pipeline is within the execution unit and provides immediate operands directly to the instruction-execution unit.

The microengine includes programmable logic arrays (PLAs) that decode the instruction or extension words. The PLAs provide microcode entry addresses for the ROM control store through the next micro-address selector. The NMA selector and the exception-control block are also part of the microengine and provide microcode branching and exception handling.

The execution-unit control section contains the decoders, state machines, and residual logic for interfacing the microengine to the execution unit. This section also decodes select fields of the instruction such as operation size or register number and provides hardware assists to the microcode.

The pipeline control/bus-interface block contains two units. The pipeline-control unit manages the instruction pipeline and controls the loop mode (discussed later). The bus-interface unit schedules and runs operand and instruction bus cycles, as well as controlling the sequencing of the bus-execution unit.

CPU architecture. The small die size allocated for the CPU compelled a cost-effective architectural implementation. Here we discuss some of the CPU design trade-offs.

One means of speeding up instruction execution is to divide the instruction unit into separate address and data-execution units that can perform arithmetic operations. From examining M68000 family designs, we determined that the parallel operation of separate units does not occur frequently when they are controlled by one microengine. Thus we determined that one instruction unit would provide an acceptable performance level.

In defining the execution unit, we investigated three primary alternatives. The first was to use one 16-bit execution unit and accept a lower level of performance for 32-bit operations. This solution not only reduced the execution unit size but also reduced performance and added to the complexity of the microengine. It also hampered the addition of 68020 instruction extensions such as 32-bit displacements or long Multiplies and Divides.

The second alternative was to employ two 16-bit closely coupled execution units, a scheme similar to that of the 68000 and 68010. The coupled arithmetic units provide for 32-bit arithmetic. The 16-bit execution unit pitch (or height) made it attractive from a layout floor-plan perspective, but the need for additional signal routing and added control complexity eliminated this solution.

We decided to use a full 32-bit execution unit, although we were initially concerned that the 32-bit pitch would interfere with the module's aspect ratio. In the modular design approach of the 68332, a size increase perpendicular to the bus direction would affect all other modules. Careful routing, placement, and design of all the cells eliminated this problem. Selection of the 32-bit execution unit provided fast instruction execution and simplified the number of control factors in a small area.

Another trade-off within the execution unit involved fast shifting. A barrel shifter can significantly increase the speed of shift and rotate instructions as well as support the bit-field instructions of the 68020. Our experience indicated, however, that the control overhead associated with a barrel shifter is quite large. For this reason, we did not implement one. Instead, we developed a shifter and controller that shifted by either 1 or 4 bits at a time, twice per microcycle (the time necessary to execute 1 microinstruction). A 13-bit shift comprises three 4-bit shifts and one 1-bit shift. This alternative produced a performance that falls between that of a dedicated barrel shifter and that of a 1-bit microcoded shifter.

An additional architectural trade-off involved the performance of Multiply and Divide instructions. The 68000 and 68010 rely on microcode branching to implement the multiply and divide algorithms. The 68020 uses additional control hardware to allow one multiply or divide step per microcycle. A multiple-bit scanning algorithm further increases multiply performance. In contrast, the 68332 uses additional control hardware to perform two multiply or divide steps per microcycle. Multiply performance compares to that of the 68020 and divide performance is significantly faster.

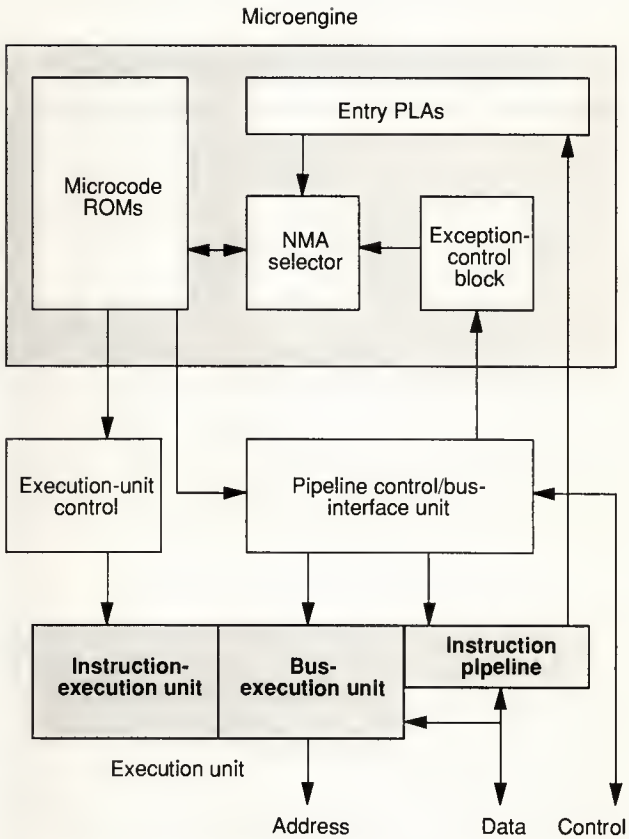


Figure 4. Block diagram of the MC68332 CPU.

Microengine. Like other M68000 family members, the 68332 has a microcoded controller, or microengine. Since processor performance is limited by both bus bandwidth and the processor's ability to use it effectively, we designed the microengine architecture to run with the internal two-clock-cycle bus. We also wrote new microcode to optimize execution speed. The microengine is pipelined into sections, which allows PLA and ROM decoding and instruction execution to each take one microcycle (120 ns at 16.67 MHz). Neither microcode nor architectural design allow fast-bus pipelining to impede performance when slower buses are used.

Simple instructions such as Add Register to Register (ADD Dn,Dn) execute in one microcycle. Multiply and Divide instructions require many microcycles to exe-

Table 3.
Sequential scheduling of instructions.

Operation	Instruction sequence 1	2	3
Bus usage			
Instruction fetch	D	E	F
Operand read	—	—	—
Operand write	—	—	—
PLA decoding	C	D	E
ROM decoding	B	C	D
Execution	A	B	C

cute. While such an instruction is executing, the next instruction remains at the PLA decoding stage as the microengine sequences through instruction execution. The CPU performs effective address (EA) calculation sequentially during instruction execution.

A separate unit could provide parallel EA calculation; however, this unit would also increase the size of the CPU. In addition, separate instruction and data buses would be necessary to effectively utilize the added performance. We traded off parallel calculation for sequential operations to obtain a smaller CPU area and fewer I/O signals. During the execution of an instruction that requires EA calculations, the PLAs first decode the microaddress for the calculation and then decode it for the instruction's basic operation. This decoding generates the addresses of the microinstructions (microaddresses) for the EA calculation and base instruction routines.

The pipeline and bus controllers schedule all stages of the microengine by using interlocking mechanisms. Table 3 shows sequential scheduling for instructions A-F (such as Add Register to Register) that each executes in 120 ns.

While instruction A is executing, the control store generates the microcode for instruction B, the PLA decodes instruction C, and the bus controller fetches instruction D. The pipeline and bus controllers schedule bus traffic. The microengine requests that reads and writes take place, but it does not need to wait for them to complete. The microengine waits when it needs the resources required for the write, the data from the reads, or additional instructions that are not yet decoded. This pipelining can result in instruction overlap, that is, the write from one instruction does not complete prior to the beginning of the next instruction.

Information affecting bus accesses is available at the earliest possible time. For example, the PLA decoding stage provides direct information to the pipeline and bus controllers that a branch instruction is about to be executed so they can adjust branching strategies. Information that a read or write is imminent is likewise provided up to one microcycle prior to address and data availability in the execution unit. The decoding PLA emits this information if the access is to occur in the first microinstruction of a sequence. If the access occurs in a later microinstruction, the previous microinstruction furnishes the information. All this information is available to the machines controlling the pipeline and the bus.

Pipeline and bus controllers. Since certain control information is available early, the pipeline controller can monitor and anticipate microcode execution and relate that with the current state of the instruction pipeline. The pipeline controller is also provided with information concerning the anticipated bus speed for the next prefetch operation. This information is used to intelligently request prefetches and allows the pipeline depth to be adjusted to maintain optimum performance for any bus speed.

The bus controller also monitors early control information and microcode execution and schedules the operand and prefetch bus cycles. The early control information is used to schedule resources so that an operand cycle can start within a clock cycle of being requested. The bus controller can also schedule up to three requests at one time, consisting of a prefetch operation and up to two operands. This facility is important for stacking operations and move memory-to-memory operations since the multiple operands can now be pipelined. Operands are the highest priority and are run in the order in which they were placed in the queue. Subsequently, prefetch results are placed in the queue on a demand basis and run whenever the bus is available. Interlock mechanisms provided within the bus controller prevent microcode from overwriting data during operand pipelining.

We combined the pipeline and bus controller functions in an attempt to minimize the time the primary microengine is forced to wait on external resources that are primarily related to the bus. The flexibility of the controllers optimizes processor performance in terms of the available bus bandwidth. As a result, on a two-clock-cycle bus the pipeline maintains a full state and provides instructions to the microengine as they are needed. On a slower bus, the pipeline depth is intentionally shorter to reduce the branch delay caused by pipeline depth. On a faster bus, the branch delay is not significant.

Loop mode. Since the 68332 has no cache (due to die-size requirements), we implemented a loop mode. It is somewhat expanded from the loop mode on the 68010 and effectively yields a 3-word cache for some instruction sequences. The loop-mode feature increases

the speed of operations such as block move, clear block, checksum block, and search block of memory. This mode is restricted to any 1-word instruction followed by the Test Condition, Decrement, and Branch instruction (DBcc) with a displacement of −4. The DBcc instruction operates on three operands: a loop counter, a branch condition, and a branch displacement. In the loop mode, the low-order word of the register specified as the loop counter is decremented by one and is compared to −1. If they are equal, the next sequential instruction executes. Otherwise, the condition code register is checked against the specified branch condition. If the condition is false, the processor branches back to an instruction that is looped. If the looped instruction causes a change of program flow, the CPU does not enter the loop mode.

Once it is in the loop mode, the processor performs only the operand bus cycles associated with the instruction and suppresses fetching instructions. Interrupts are still allowed during the loop mode and, if taken, result in the CPU exiting this mode. Table 4 is an example of a block move performed using the loop mode.

Table 5 shows the average number of clock cycles per move. The DMA column refers to the amount of time necessary for a direct-memory access that requires a separate read and write per move (dual-address DMA). The in-line column shows the number of clock cycles if the code contains 100 move instructions and no branching. The loop mode helps increase performance the most on slower buses because it removes extra instruction fetching.

The fast two-clock-cycle bus increased performance and allowed the use of the 16-bit data bus without sacrificing high performance. On MCUs, the pin count limits the number of peripherals and I/O ports. MCUs are more cost sensitive than microprocessor units, a factor that also lowers the number of pins available in a package. A 32-bit external bus would not leave enough pins for the planned peripherals. Even without the external two-clock-cycle bus, the internal RAM (and future ROM) can be accessed in two clock cycles. Frequently used variables and subroutines can be copied into internal RAM. In addition, internal RAM can be used for stacking, which results in faster context switching.

CPU performance. We optimized the Multiply, Divide, Table, and Shift instructions for computationally intensive applications. See Table 6 for a list of execution times (in clock cycles) for those instructions.

The loop mode increases the speed of block moves or string searches. Context switching is significantly faster than in the 68010 due to the faster stacking on the two-clock-cycle bus. When it uses slower memories, the processor can track bus speed and intelligently adjust its performance to the available bus bandwidth. Most register-to-register instructions execute in two clock cycles (one microcycle, one bus cycle, and one instruction cycle). Thus, performance peaks at 8.4 million instructions per second (MIPS) at 16.77 MHz. (A typical program would not sustain this level.) A 16.77-MHz 68332 operating on a 16-bit/two-clock-cycle bus can perform at 80 percent of the speed of a

Table 4.
Block move using the loop mode.

Label	Instruction	Operand
	moveq	#100-1,d0
loop	move.l	(a0)+,(a1)+
	dbra	d0,loop

Table 5.
Average clock cycles per transfer.

Bus speed	DMA	In-line code	Loop mode	No loop mode
2	8	10	13.07	17
3	12	15	15.11	21
4	16	20	17.20	28
5	20	25	20.40	40

Table 6.
List of instruction-execution times.

Mnemonic	Function	Clock cycles
ADD.L Dn,Dn	Add	2
OR.L Dn,Dn	Or	2
MOVE.L Dn,Dn	Move	2
CMP.L Dn,Dn	Compare	2
MUL 16 × 16	Multiply	26
DIVU 32 × 16	Divide unsigned	32
DIVS 32 × 16	Divide signed	42
ROL #n,Dn	Rotate left	6
LSL Dm,Dn	Logical shift left	
	1-6, 8, 12 bits	6
	63 bits	22
TBLS <ea>,Dn	Signed table lookup	39

68020 operating on a 32-bit/three-clock-cycle bus without a cache. With a 16-bit/three-clock-cycle bus, the CPU can achieve about 65 percent of a 68020's 32-bit/three-clock-cycle performance. (See accompanying box for the Table instruction.)

The Table Instruction

The Table Lookup and Interpolate instruction supports both a rounded and an unrounded result. Both variants can provide a byte, word, or long result. They also furnish two formats for the interpolation data: an *n*-element table stored in memory and a two-element "table" stored in a pair of data registers. The latter form provides a means of calculating a surface interpolation. Figure A is an example of this calculation with a 257-word table.

In this example, the table consists of 257 I-word entries. The function is a straight line within the range of $32768 \leq X \leq 49152$ as shown on the plot. Table A demonstrates some table entries for this example.

For this example, the Table instruction is executed to look up the value for $X = 41856$. The upper 8 bits generate the table entry offset of 163, and the lower 8 bits generate the interpolation fraction of 128. Using this information, the instruction calculates the dependent variable Y.

$$Y = 1669 + [128(1679 - 1669)] \div 256 = 1674$$

For highly linear functions, the data can be compressed into a smaller table. For example, Table A can be compressed into a five-entry table by limiting the range of X (seen by the Table instruction) from 0 to 1023 (Table B). Prior to the Table instruction, X must be scaled. In this case, the scaling factor is 64; the scaling is done by the Logical Shift Right (by 6 bits) instruction (LSR.W #6,Dx).

For the same value of X—41856—the number is scaled to 654. The upper 8 bits generate the table entry offset of 2, and the lower 8 bits generate the interpolation fraction of 142. Using this information, the Table instruction calculates the variable Y.

$$Y = 1311 + [142(1966 - 1311)] \div 256 = 1674$$

Note that the chosen function was linear between the points entered into the table. Had another function been chosen, the interpolated values for Y may not have been identical.

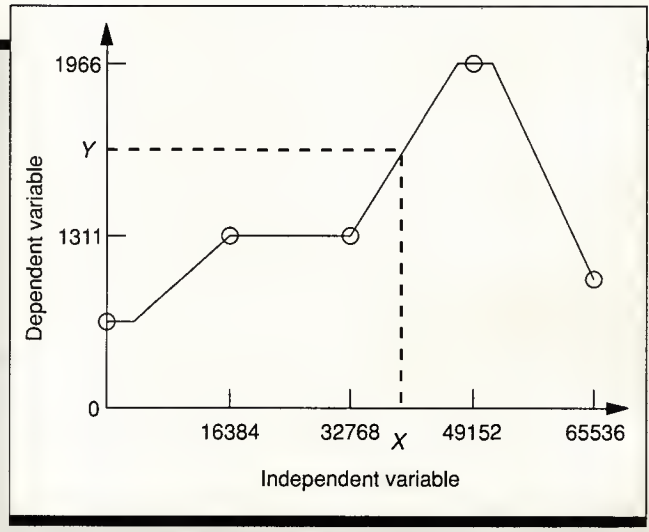


Figure A. The Table instruction calculation.

Table A.
Table-entry examples.

Table entry number	X value	Y value
128	32768	1311
—	—	—
162	41472	1659
163	41728	1669
164	41984	1679
165	42240	1690
—	—	—
192	49152	1966

Table B.
Compressed table.

Table entry number	X value	Scaled X value	Y value
2	32768	512	1311
3	49152	768	1966

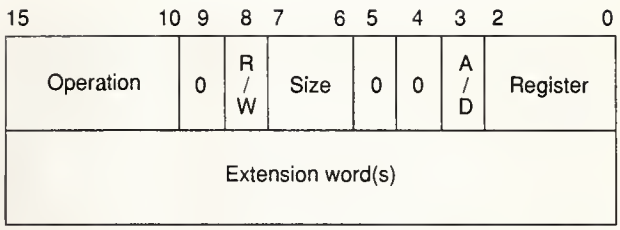
Development support. We added several unique features to the CPU to aid in the debugging of programs. Developers typically produce prototypes and debug initial code by using a debugging monitor in ROM as well as a serial line that communicates with a terminal or host processor. Background mode effectively gives the developer the same support in microcode. This support is transparent to the hardware and software of the application. Consequently, prototype hardware differs less from the final design. This procedure also allows for production-hardware and field debugging.

The background debugging mode uses a three-wire, bi-directional, serial interface (similar to the SPI on the MC68HC11 and the SPI section of the QSM) to directly interface with the CPU. Debugging commands proceed through the serial interface to examine and/or modify memory and registers, branch to a code patch, or return to normal program execution. Figure 5 depicts the background command format, while Table 7 summarizes the associated commands.

The system enters background mode by executing the background instruction or asserting the hardware breakpoint line. A double bus fault has the same effect. To prevent accidental entry, background mode can be disabled at reset.

Hardware breakpoint is another new development feature. When a bus cycle runs, an external breakpoint pin is asserted to tag the bus cycle. When that data is used, a breakpoint occurs. At the next instruction boundary, either a breakpoint exception is taken or background mode is entered. Hardware breakpoints allow a simple comparator to stop program execution at a particular spot during debugging.

Pipelined architectures such as that of the 68332 are inherently difficult to trace in a coherent fashion. Without internal state information, opcode tracking is difficult at best—and sometimes impossible. Typical pipelined microprocessors provide no indication of which instructions are executed and which are flushed from the pipeline before they are executed.



A/D	Address or data register
Operation	Particular background mode command
Register	One of eight address or data registers
R/W	Read or write operation
Size	Byte, word, or long operand size

Figure 5. Background command format.

Table 7.
Command summary.

Operation	Mnemonic	Description
Read register	RDREG	Reads data register
	RAREG	Reads address register
Write register	WDREG	Writes data register
	WAREG	Writes address register
Read system registers	RSREG	Reads PC, SR, USP, SSP, SFC, DFC, and VBR registers
Write system registers	WSREG	Writes PC, SR, USP, SSP, SFC, DFC, and VBR registers
Read memory location	READ	Reads byte, word, or long word from memory
Dump memory block	DUMP	Reads the next memory location and increments
Write memory location	WRITE	Writes byte, word, or long word from memory
Fill memory block	FILL	Writes the next memory location and increments
Resume execution	GO	Exit background mode
Patch code	CALL	Branch to subroutine and go
Reset peripherals	RST	Equivalent to Reset instruction
No operation	NOP	Null command

Function code outputs are augmented in the 68332 by two supplementary signals to allow the simulation of the internal instruction pipeline. Instruction Pipe (IPIPE) indicates the start of each new instruction and mid-instruction pipeline advance. Instruction Fetch (IFETCH) indicates those bus cycles during which the operand will be loaded into the instruction pipeline. Pipeline flushes are also signalled with IFETCH. Monitoring these two signals deterministically signals all pipeline flushes and permits an analyzer to synchronize itself to the instruction stream.

The 68332 also provides for visibility of internal accesses. During normal operation the internal and external buses are only coupled during external accesses. This procedure allows internal activity to continue while the external bus is granted away to another bus master for improved performance. The external bus interface of the 68332 can be programmed to provide internal access visibility via a special "show cycle." For debugging, the external bus interface can also be programmed to always couple the internal and external buses; when the external bus has been granted away, internal bus operations cease.

Time processor unit

As more high-performance micros are used in control applications, users expect a great deal from their timing systems. Users need to attack problems that require more flexibility, higher frequency, and finer resolution timing control. However, the number-one constraint of micros in these applications is the inability to perform high-frequency timing functions such as high-performance engine-control strategy. CPU overhead associated with servicing the timer system and other peripherals limits high-frequency timing. To a lesser extent, this servicing in a general-purpose CPU timer system results in longer service routines because the instruction set has not been optimized for timing tasks.

Multipurpose micros must accommodate both simple and complex timing tasks. Designers have commonly designated timer pins for the operations of input capture and/or output compare. They added specialized hardware for higher frequency timing tasks such as pulse accumulation and pulse-width modulation. Users with different needs were forced to add timer peripherals, build them with gate arrays, or maybe use an application-specific integrated circuit. These procedures increase system costs and design-cycle times.

We evolved the time processor unit (TPU) to solve these problems in control applications. The TPU performs a number of timing tasks as a peripheral device.

Overview of the TPU. Because it is a microcontroller, the TPU can perform timing tasks without CPU intervention. Consequently, CPU overhead does not constrain high-frequency timing tasks. An instruction set tailored specifically for timing tasks and an instruction cycle of two system clocks (120 ns at 16.67 MHz) reduces the time required to process timing events.

Any TPU time function in the instruction control store can be programmed to operate on any of the 16 TPU pins. A time function can be used on multiple channels. Users can mix timer pin usage and define new functions through emulation.

We plan to provide the TPU in several preprogrammed versions that contain up to 16 time functions for real-time applications. The first version contains time functions that range in performance from simple digital I/O to complex angle-based automotive engine control. Other functions include

- stepper motor control,
- pulse-width modulation,
- frequency measurement,
- high time accumulation,
- frequency divide/multiply,
- pulse accumulator,
- output compare, and
- input capture.

Time functions associated with some pins can be of a higher frequency than for other pins. TPU servicing priority can be allocated to each pin for specific applications. The TPU scheduler manages servicing of the pins and ensures worst case latency calculations. Establishing worst case latency for timer pins is critical to timer-system integrity.

The timer-channel hardware associated with each pin can be configured in multiple ways to facilitate a wide variety of time functions. Timer-pin outputs or inputs can be synchronized to one or both TPU timer count registers (TCRs). One TCR is clocked internally, while the other can be selectively clocked externally as well. The TCRs can be clocked at a maximum frequency of once every four system clocks, which yields a resolution of 240 nanoseconds at 16.67 MHz.

A TCR is commonly referred to as a time base. Some applications need to associate a position time base to a real-time time base. For matching output, each channel is equipped with a greater-than or equal-to comparator to guarantee operation. For capturing input, the initial or last occurrence of the proper pin transition can be synchronized to a time base, regardless of the frequency of the input.

Complex timing tasks can require multiple pins to work in concert. The feedback loop for complex timing tasks has been implemented on some microcontrollers that use a CPU. Consequently, CPU latency and servicing inefficiencies constrain microcomputer applications in complex control systems. The TPU facilitates expedient and deterministic response for timing events that affect the operation of multiple channels. The 68332 can consequently solve more complex timing tasks than other microcontrollers.

Since the CPU and TPU operate in parallel, coherent (age-identical or logically related) data access by either must be ensured. Furthermore, the number of operands accessed coherently may vary depending upon the application.

Here we describe the TPU architecture in detail, discussing both the internal organization of the device and its external interfaces. We also explain how the architecture facilitates high-frequency timing, flexibility, and operand coherency.

TPU architecture. The internal structure of the TPU as shown in Figure 6 consists of:

- timer channels and associated pins,
- a microengine,
- a scheduler, and
- the host interface.

The overall architecture of the device is RISC-like in the sense that there are no expanded instructions. All instructions execute in one instruction cycle, which consists of two system clock cycles. Device is service-request driven rather than interrupt driven. The scheduler updates a channel register with the number of the channel next to receive service. The output of the channel register is decoded to arrange context switching to the memory-mapped facilities associated with the channel that is granted service.

Servicing a pin commences with the execution of instructions that pertain to the time function programmed for that pin. Since complex time functions can contain multiple conditional flows or phases, a direct branch based on channel flags is initially performed to reduce software overhead.

The major features of the TPU are as follows.

- It contains 16 I/O pins. Each pin is associated with a unique timer channel.
- Each channel can perform any time function.
- Each channel has an event register consisting of a capture register, a match register, and a greater-than or equal-to comparator, all 16 bits.
- Each channel can be synchronized to one or both of the two 16-bit, free-running timer count registers TCR1 and TCR2. Each channel pin can resolve to the system clock divided by four.
- Register TCR1 is clocked from the output of a programmable prescaler whose input is the system clock.
- Register TCR2 is clocked from the output of a programmable prescaler whose input is the external TCR2 pin. TCR2 may be used as a hardware pulse accumulator clocked from the external TCR2 pin or as a gated pulse accumulator of the clock that increments TCR1.
- All pins have at least six 16-bit, time-function operands known as parameter registers that are contained in dual-access RAM accessible from both the TPU and CPU.
- A scheduler with three priority levels segregates high-, middle-, and low-priority time functions. Any channel may be assigned to one of these priority levels.
- Worst case latency for the servicing of any channel is deterministic.
- All time functions are programmed in an instruction control store or microcode ROM.
- Emulation and development support can create and debug new time functions. Features such as breakpoint, freeze, and single step give internal register accessibility.
- The device accommodates coherent transfers for *n* parameters.

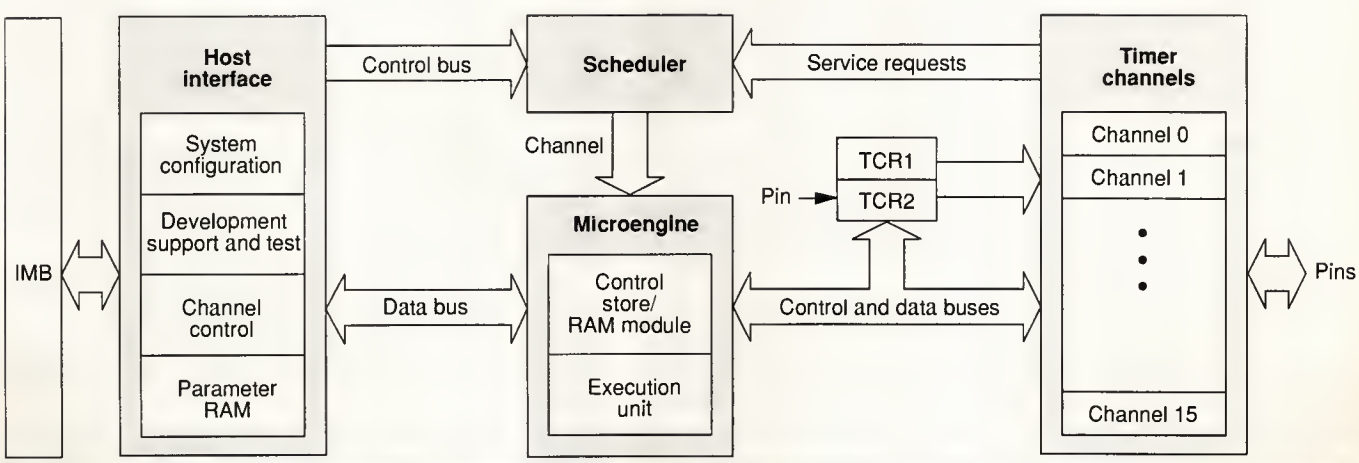


Figure 6. Block diagram of the MC68332 TPU.

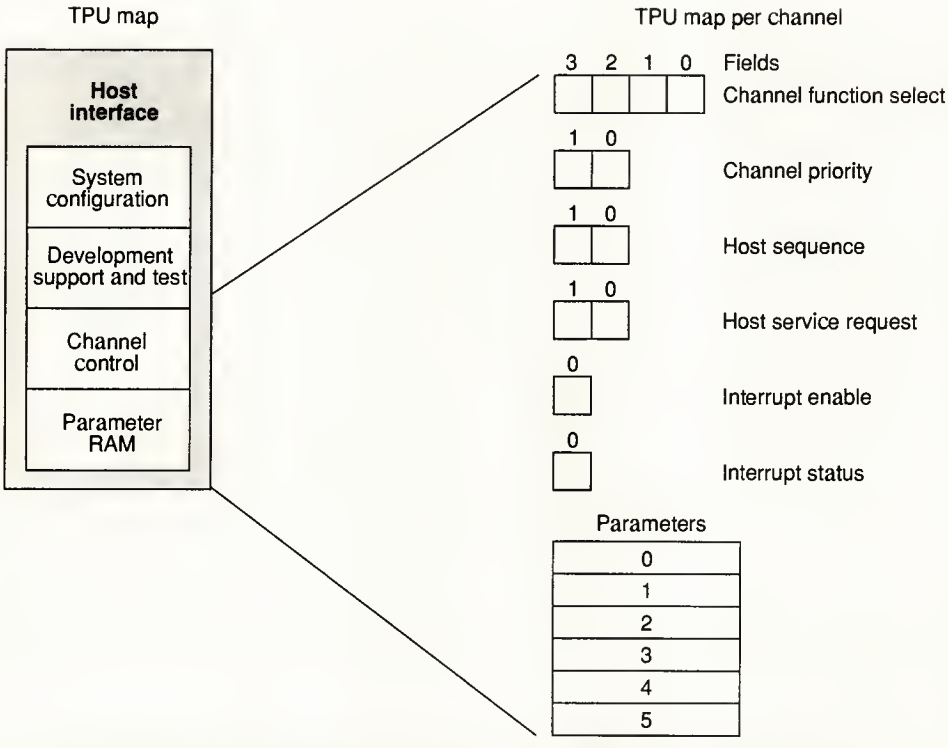


Figure 7. User's program model.

Figure 7 shows a user's program model on a per-channel basis.

Timer channels. All channels perform the two most primitive operations of the TPU: matching and capturing events. A match event occurs when a specified TCR increments to the value stored in the match register (see Figure 8), or is greater than the match register value. A capture event occurs when a specified TCR is loaded into the capture register. These events relate to the external world via the pin control. The configuration of the latch control allows several types of channel operation. Studying the three sections of a channel, as shown in Figure 8, clarifies how events are performed and used.

The event control consists of an event register and the control logic necessary to govern event register operation. The event register consists of the comparator and the match and capture registers. It interfaces to the microengine via the TPU bus and to TCR1 and TCR2 through their buses.

The fact that the TCR1 and TCR2 buses traverse all channels facilitates simultaneity of events. A capture event can be initiated either by (a) a match event that is recognized by the assertion of the match-recognition latch (MRL) or (b) an input transition that is detected at a channel pin by the assertion of the transition-detection latch (TDL). TCR1 and TCR2 can be nonexclusively programmed to associate with either match or capture events. Users can synchronize an event on one time base with the associated value of a different time base, such as matching on time base TCR1 and capturing on time base TCR2.

The latch control retains information concerning the events on a channel. This section also issues service requests to the scheduler as a result of either match recognition or input transition detection. The microengine accesses the latch control during a channel's service time to determine the state of the channel and to control events and the negation of certain latches within a channel.

Two channel latches, the MRL and the TDL, record match events and capture events. It is important to distinguish between a match event and match recognition. Because each channel employs a greater-than-or-equal-to comparator, many match events occur when the specified time base exceeds the value of the match register. However, once a recognized match event asserts the MRL, further match events are prevented by the negation of the MRL enable. The channel match register is written with a new value that schedules another match event. This mechanism prevents the recognition of inadvertent match events.

It is only through the assertion of the MRL that a match event becomes active at the pin or a service request is issued to the scheduler. In addition, if a channel is being serviced due to a condition other than a match, users can selectively disable match events during service. Once service begins, users can ensure the cancellation of a pending match event as well as the rescheduling of another one. This feature eliminates a major disadvantage of early FIFO-based timer architectures—the inability to easily cancel a previously scheduled match event.

The pin control is the hardware through which timer events are translated into or interpreted from a specified action at the pin. As an output, a pin can respond to a match event to force the pin to a high, low, or toggle position.

As an output, the pin can be directly forced to high or low without the necessity of a match event. To affect a capture event, an input pin can respond to a rising or falling edge—or to both.

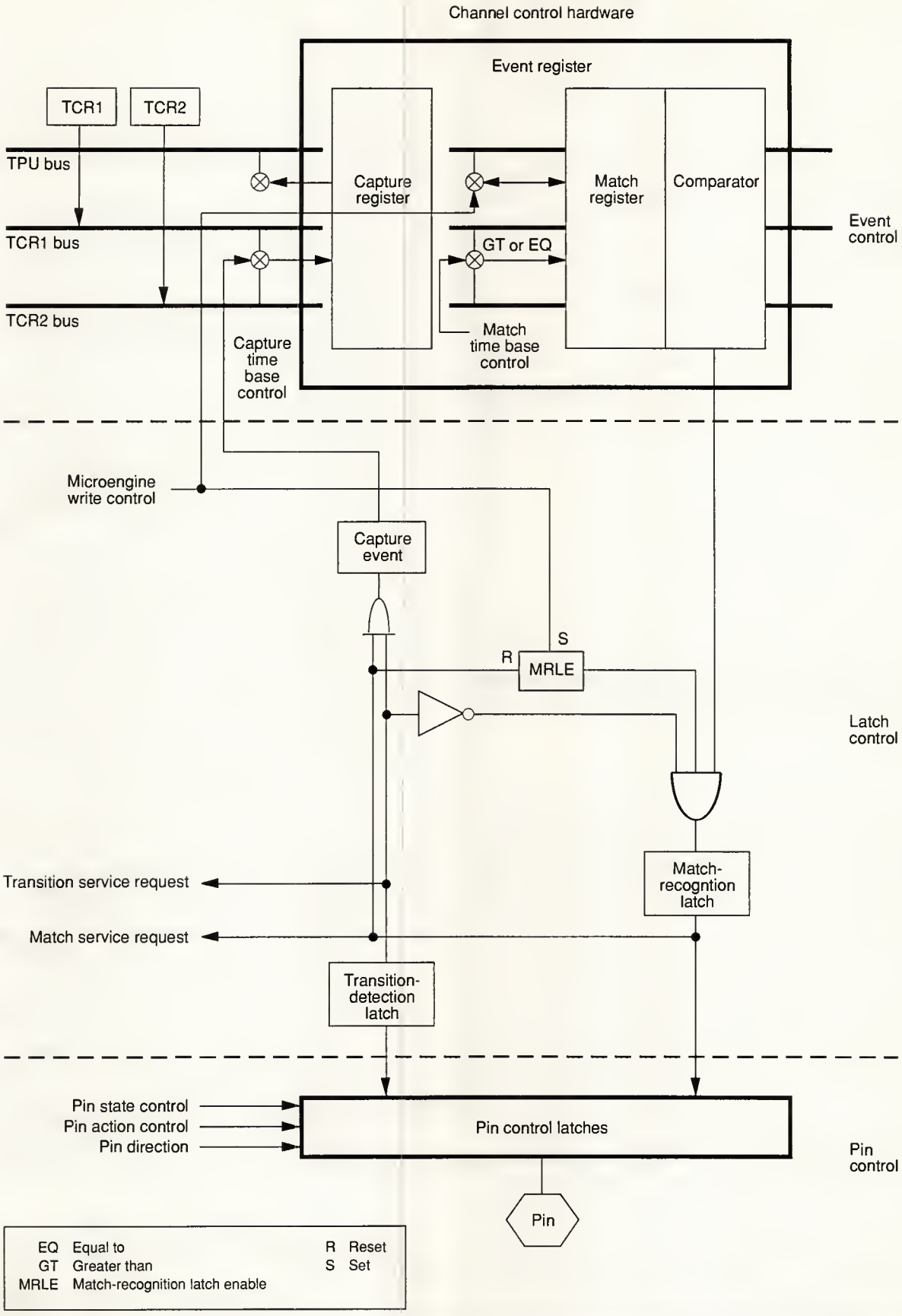


Figure 8. A TPU channel.

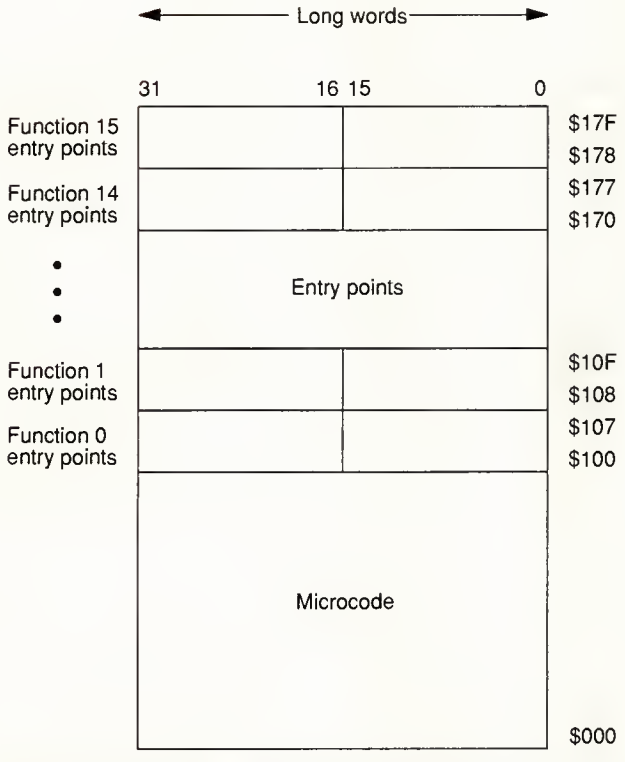


Figure 9. Control store organization.

In an effort to ensure clean signals, all input signals presented to a channel pin are fed through a hysteretic circuit that drives a synchronizer. The output from the synchronizer feeds a digital filter. This filter blocks transitions shorter than two system clock cycles and passes transitions longer than four system clock cycles.

TPU microengine. The microengine provides control over the execution unit, channels, and access to parameter RAM for time-function synthesis. It acts on demand, switching channel context to channels that are currently requesting service. When no requests exist, the microengine continues to run, executing NOP instructions.

The TPU microengine implements a pipeline in which instruction decoding and execution overlap instruction fetching. Because the TPU is a RISC-like machine that employs simple instructions, instruction decoding takes only one half of a clock cycle. In addition, the TPU can sustain an execution rate of greater than 8 native MIPS at an operating speed of 16.67 MHz.

A 9-bit microinstruction program provides sequential access to the control store. Control store words are 32 bits in length. A branch condition or an entry-point selection alters sequential access of the control store. Contents of a selected entry point that contains a beginning control-store address or the contents of the return address register (RAR) have the same effect. The RAR saves the return address where execution resumes after subroutine completion. The RAR supports one level of subroutine nesting.

The microengine supports several other sequencing features like the repetitive, programmable execution of one microinstruction for up to 17 times. This feature supports fractional scaling as well. Any sequence of up to 16 microinstructions can serve as a subroutine without using a Return from Subroutine as the last instruction of the subroutine. The 4-bit decrementer in the execution unit implements this hardware return.

The microengine also contains two 16-bit flag registers. Each register associates one flag with each channel, for a total of two flags per channel. The channel flags are varied under microcode control and retain state information that helps direct control-store execution flow.

In addition, the microengine supports absolute and relative addressing. Absolute address calculation indicates that the operand is the address. Relative address calculation indicates that the operand field relates to the channel number. The TPU microcoder uses relative address calculation to design reentrant microinstruction sequences, that is, sequences that can execute on any channel. Operands for both direct and relative addressing include parameters, channel number, link channel, and decrement count values. A channel number is a 4-bit encoded value that represents one of 16 channels. When this value is written to the channel number register, channel-specific attributes are accessible.

The control store associated with the microengine is organized as 384 locations that each contain 32 bits. The top portion of the control store map contains entry points organized as 16 blocks of 16 entry points that each contain 16 bits, while the bottom portion of the map contains microcode (see Figure 9). The number of time functions implemented in the control store can vary from one to 16. Because of this variance, the microcode address space expands to include unused entry points.

When a channel is scheduled for and granted service, the microengine fetches one of the 16 entry points related to the selected time function. The specific entry point that is fetched is a function of a channel's state. An entry point contains information about the sequence of microinstructions to be executed. This information includes the fields for

- the beginning of microcode addressing,
- a preload destination register in the execution unit (see Figure 10),
- a match enable that selectively enables matches during channel service, and
- a channel parameter preload source that provides data to the preload destination register in the execution unit.

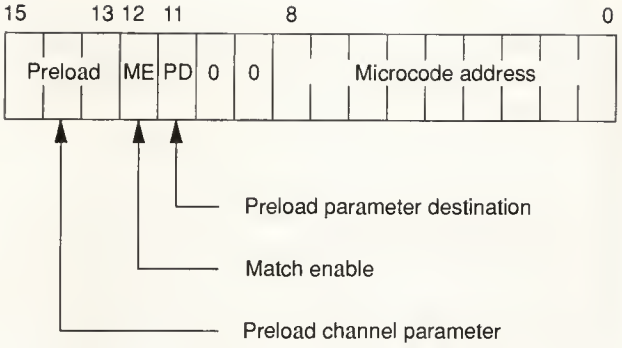


Figure 10. Entry point format.

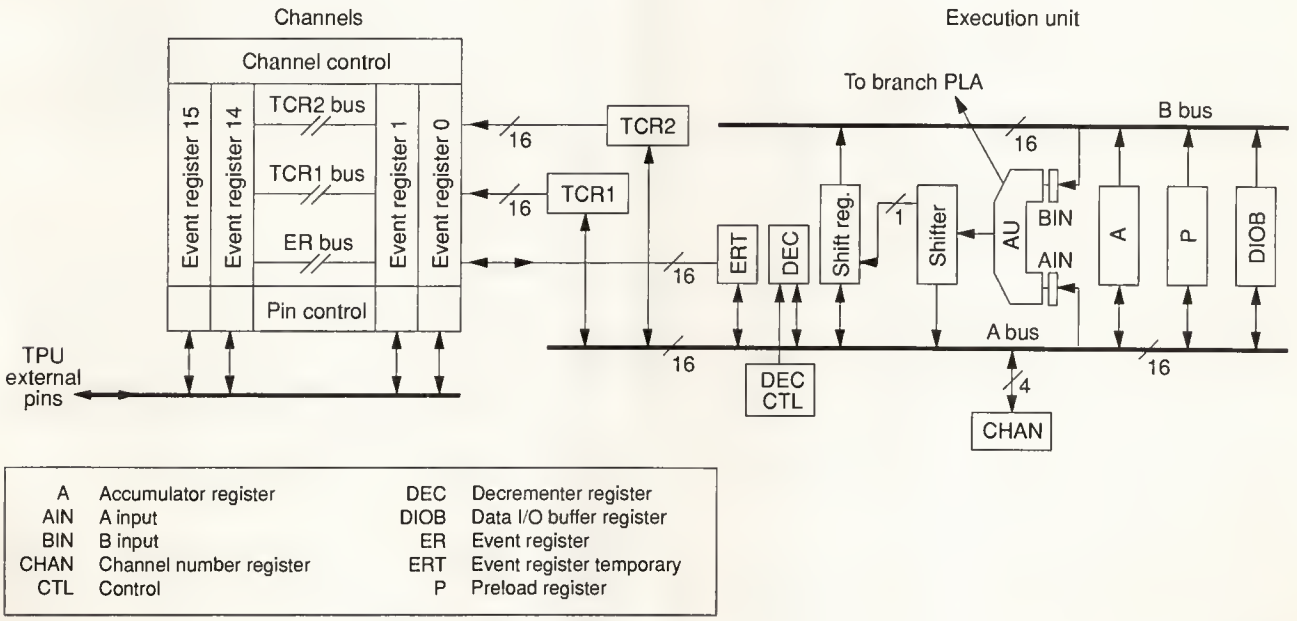
These fields are used during channel context switching. We chose this indirect method of obtaining the beginning address of microcode execution so that microcode could be placed anywhere in the memory map.

The execution unit consists of the arithmetic unit (AU), functional units, general-purpose data registers, and specialized registers that can also function as general-purpose data registers (see Figure 11). The registers and functional units within the execution unit are

- a 4-bit channel number,
- a 4-bit decrementer,
- the 16-bit AU,
- a 16-bit shift register,
- a 16 × 1-bit, shift-and-rotate shifter,
- a 16-bit preload register,
- a 16-bit data I/O buffer,
- a 16-bit accumulator, and
- a 16-bit event register,

The execution unit supports word, byte, and nibble operations at various levels. The AU can perform add and subtract operations for 8- or 16-bit operands. The shift register accomplishes shift left, shift right, and rotate right logic operations. The microengine can also implement a $16 \times n$-bit ($1 \leq n \leq 16$) fractional scaling by means of the shifter and shift register. Each additional bit of scaling can execute within two clock cycles. A 16×16-bit fractional scaling executes in 1.92 microseconds at a 16.67-MHz clock frequency.

A need for coherent data occurs at the microengine/channels interface. For data to be coherent, all data must be updated before any of it is read. Conversely, coherency can require that all data must be read before any of it is updated. Coherency problems occur whenever data must be shared among modules and submodules that operate asynchronously with one another. We addressed the coherency problems at the microengine/channel interface in the following manner.



A	Accumulator register	DEC	Decrementer register
AIN	A input	DIOB	Data I/O buffer register
BIN	B input	ER	Event register
CHAN	Channel number register	ERT	Event register temporary
CTL	Control	P	Preload register

Figure 11. Execution unit and channel control.

Because the TPU is a real-time machine, the state of the channels can change asynchronously with respect to the operation of the TPU microengine. When a requesting channel is granted service, the state of that channel's match and capture registers must contain correlated data with respect to the state of the channel's TDL and that of the MRL. The same is true for the channel pin state (CPS) that is latched at the branch PLA. This correlation is essential because the TDL, MRL, and CPS determine next-state information (where the next microcode execution begins) and the contents of the match and capture registers must contain data to reflect the entered state. Consequently, before channel service begins, a "snapshot" of the channel state is saved at the branch PLA. That saved state must be coherent if the microengine is to make accurate calculations based on the current state and deterministically predict the next state.

Consider the case in which the match register is configured to capture one time base TCR1, and the capture register is configured to capture the other time base TCR2. If a match occurs nearly coincident with a snapshot taken of the channel state, then either one of two things must occur to retain coherent data:

1) If the match occurs *before* the snapshot of the channel state, the associated capture of TCR2 into the capture register initiated by the match must complete before the channel state is latched. The branch PLA records that the MRL is asserted; the capture register contains the corresponding new TCR2 data.

2) If the match occurs *after* the snapshot of the channel state, the branch PLA records a negation of the MRL. The capture register contains the old data.

The TPU module provides a high degree of interchannel communication, that is, the ability to reference or synchronize one channel's operation to specified action(s) on another channel. Interchannel communication can occur without CPU intervention. The TPU employs two types of this communication: direct and requested. The first is provided through the use of the change-channel mechanism. Writing a value to the channel-number register during channel service effectively changes the channel context. This mechanism lets any TPU channel operate on another channel state.

Requested interchannel communication is accomplished through a link service request, that is, a signal from a source channel to the scheduler to service the destination channel of the link service request. The interpretation of the link signal is determined by the destination channel during channel service by the microcode executing on the channel.

The emulation bit in the module configuration register provides emulation through the use of the RAM module. This module acts as the instruction-control store for the TPU. When the TPU is in emulation mode, an auxiliary bus connects the RAM and TPU modules. Access to the RAM module via the intermodule bus is disabled. A 9-bit address bus, a 32-bit data bus, and control lines allow data transfer between the modules. RAM module access timing matches that for the TPU ROM to ensure exact emulation.

Scheduler. The TPU scheduler functions as a real-time executive implemented in hardware. The executive allocates microengine time on a channel-demand basis, as further described. One of four service-request sources initiates a request for service on a per-channel basis. As previously discussed, the two sources initiated in the channel hardware are match recognition and transition detection. The third service request is initiated under microcode control by writing a link service request to the link register. The host CPU initiates the fourth service request by writing a host service request field associated with the channel.

Because the relative frequency of events can vary depending upon the application a channel is required to perform, the scheduler provides

- an orderly method for servicing requesting channels that ensures no channel can be blocked from receiving service,
- the relative frequency of channel service to be programmed, and
- the worst case latency of event servicing that can be calculated (deterministic).

The scheduler employs a rotating service-allocation queue to determine channel-service frequency. It contains three priority levels and seven service time slots. The scheduler can assign any channel to a high-, mid-, or low-priority level. Four time slots are allocated for high-priority channels, two for mid-priority channels, and the remaining service time is allocated to low-priority channels. If channels do not need the time slot, the scheduler grants that slot to another priority level based on the following order:

high → mid → low
mid → high → low
low → high → mid.

The scheduler does not waste time if the assigned priority level cannot use the service time slot. Multiple channels requesting simultaneous service on the same priority level are serviced in a round-robin fashion beginning with the lowest numbered channel that requests service. Any pending service request on the same priority level is granted prior to a new request from a channel that has already been serviced.

Host interface. The host-interface registers (see Figure 6) can be partitioned into four classes:

- system configuration,
- channel control,
- parameter RAM, and
- development support and test.

The system-configuration registers affect the operation of the TPU as a whole and are not channel specific. Two such registers—module and interrupt configuration—control

- the selection of emulation or low-power stop modes for the TPU,
- the division of the clocking source for time bases TCR1 and TCR2,
- the interrupt-arbitration number of the TPU subsystem with respect to other subsystems resident on the 68332 MCU that generate interrupts, and
- the interrupt-request level of TPU channels.

The channel-control registers configure operation on an individual channel basis. The channel function-select registers assign a time function to a channel, while the channel priority registers assign a priority level to a channel. The host service-request registers—when written to by the host CPU—issue one of three kinds of host service requests, and host sequence registers select the operation mode of a time function.

The parameter registers constitute a 100-word RAM workspace through which the host CPU and the TPU communicate. The host CPU or the TPU can dually access the parameter registers one register at a time. From the TPU perspective, the parameter registers are organized as 6 words associated with channels 0 through 13 and 8 words associated with channels 14 and 15. When channel context changes (such as when a channel receives service), an associated parameter register context switch occurs as well. From the CPU's perspective, the parameter registers are organized as a continuous block in which there are two nonimplemented word locations every 6 words except for the last 16 words, which contain no holes.

The CPU uses parameter registers to control certain characteristics of the time function that operates on a channel. The CPU writes data into the appropriate parameters, which are read by the TPU. This data includes the period and/or high-time of a pulse-width modulation time function. Likewise, the CPU can read certain data (such as the period of a time function) from parameter(s) that the TPU calculates and writes to other appropriate parameter(s).

The need for coherent data also occurs at the CPU/TPU interface. Both the CPU and TPU modules can read and write parameter RAM in an asynchronous manner. Because the parameter RAM is dual access, access collisions and coherency problems occur. The most common is a lack of 2-word parameter coherency. As such, two-operand coherency is supported via hardware in the arbitration logic that governs access to the parameter RAM. Long word accesses (back-to-back IMB cycles) by the CPU are coherent, as is every successive pair of TPU accesses. From the TPU perspective, the microcode must be written by means of successive RAM accesses to produce coherent two-operand pairs.

The TPU scheduler functions as a real-time executive implemented in hardware.

Allocating a portion of RAM to act as coherent data registers (CDRs) accomplishes multiple-operand coherency. A special microcode routine and a predefined protocol interlock the data to ensure coherency. Because the TPU microcode routine moves the data to and from the CDRs, no read/write collision can occur if the host CPU follows the protocol for accessing CDRs. A semaphore flag is implemented via microcode to allow multiple processes to use the CDRs.

The test register provides a means to configure and control the module for test purposes. Once the 68332 is in test mode, certain serial scan paths can be configured to allow certain registers to be scanned. Scannable registers include the micro program counter, microinstruction register, branch PLA, micro program counter breakpoint register, channel breakpoint register, and scheduler PLA. Access to certain registers associated with scheduler operation is also available to the host CPU. In addition, the TPU module can be configured for one-step operation. In single-step operation, the TPU reaches a halted state after each microcycle executes to allow the host to examine the state of the TPU.

The development support registers enable microcode development and debugging. The four classes of registers are

- micro program counter breakpoint,
- channel breakpoint,
- control, and
- status.

The micro program counter and channel breakpoint registers, in conjunction with the control register, control the starting and stopping of the TPU microengine. The host CPU can set a breakpoint to halt the TPU microengine as a result of various combinations of channel service requests, the channel number scheduled for service, and the micro program counter address. The channel-service request breakpoint occurs under the following conditions:

- a TDL assertion,
- an MRL assertion,
- a link-service request, or
- a host-service request.

The same is true if (a) the scheduled channel number matches the channel breakpoint register or (b) a micro

program counter is loaded during channel service with an address that matches the contents of its breakpoint register.

Whenever the configured breakpoint condition is detected, the system asserts a corresponding status flag and halts the microengine.

As requirements increase and technology allows, Motorola plans to include higher performance CPU modules in the 68300 family. These CPUs—together with the existing peripheral modules—promise to generate higher performance devices.

Designers can independently create peripheral modules and integrate them with existing modules to generate a new microcontroller for a specific purpose or application.

Joe Jelemensky is a principal staff member of the Motorola Microcontroller division in Austin, Texas. He was the systems design project leader for the MC68332 and participated in the architectural definition and specification of the microcontroller. His current research interest is in serial communications architectures for local area networks.

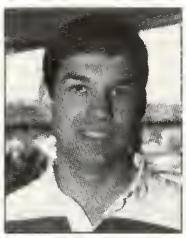
Jelemensky received his BSE degree from the University of South Florida and is a member of the IEEE.

Vernon Goler is a principal staff member of the Motorola Microcontroller division. He participated in the architectural design and specification of the MC68332. His current research interest is in timer architectures for real-time control.

Goler received his BSEE degree from Purdue University and is currently working on an MSEE from the University of Texas. He is a member of Eta Kappa Nu and Sigma Pi Sigma.

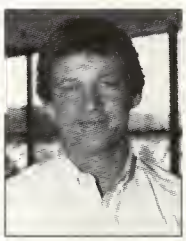
Brad Burgess is a senior member of the Motorola Microcontroller design organization. He participated in the architectural design of the CPU and the intermodule bus for the MC68332.

Burgess received his BSEE and MSEE degrees from Texas A&M University. He is a member of Eta Kappa Nu and the IEEE Computer Society.



James Eifert is a senior member of the Motorola Microcontroller design organization. He was involved with the architectural design and specification of the MC68332 CPU.

Eifert received his BE degree in electrical engineering and computer science from Vanderbilt University. He received his MS degree in computer engineering from Carnegie Mellon University. He is a member of the IEEE Computer Society.



Gary Miller is a senior designer with Motorola in the Microprocessor Products Group. His current interests include microprocessor and peripheral architectures and very large system integration design.

Miller received the BSEE degree from the University of Houston.

Questions regarding this article may be directed to Joe Jelemensky, Motorola Inc., M/S OE320, 6501 William Cannon Drive West, Austin, TX 78735-8598.

A Comparison of RISC Architectures

Evaluating the newest chips for your needs can take some time and thought. Here's help in deciding what's important to consider.

Richard S. Piepho
William S. Wu

AT&T Bell Laboratories

Numerous new RISC architectures have appeared in the marketplace over the past few months. Among these are the Intel i860, the Motorola 88000, and the Sun Microsystems Sparc architectures. Each claims great performance increases over the existing CISC architectures and superior performance and features over their RISC rivals.

Here, we compare and analyze the relative strengths and weaknesses of the three architectures in a number of key architectural areas. Based on this comparison, we assess their advantages and disadvantages. We seek to determine whether one architecture is clearly superior or inferior in the long term because of sufficient advantages or disadvantages it possesses over the others.

First we discuss the relative importance of an architectural comparison, as opposed to a comparison of implementations, and follow it with a high-level overview of each of the architectures. We examine, in detail, each of the architectures on a number of key architectural areas. Finally, we summarize the overall relative strengths and weaknesses of each architecture. (We also explain some of the specialized architectural vocabulary in the accompanying Definition of Terms.)

Architecture vs. implementation

Various proposals for drawing the line between computer architecture and computer implementation have existed since the term "computer architecture" was first used in the description of the IBM System/360. The original strict definition proposed by Blaauw[1] limited the architecture to just the instruction set and execution model. All else makes up the implementation. A more-encompassing definition proposed by Stone[2] sets the architecture as the "instruction set and structure down to the functional modules" of the system. Various other definitions fall between these two extremes.

For our purposes, however, we adopt the original strict definition. We define the architecture as only software-visible features—including the basic instruction set and memory management architectures. It does not include the specification of the functional modules used to implement these features.

0272-1732/89/0800-0051 © 1989 IEEE

Definition of Terms

ABI, or Unix System V application binary interface for a CPU architecture, defines a "binary" system interface standard. This standard supports compiled application programs running on computer systems that are based on the same CPU architecture.

An **atomic** instruction retains exclusive use of a flag (for example, a semaphore) through completion of the instruction cycle. Exclusive use of a flag prevents the flag from being modified while the instruction operates on it.

Byte-ordering or addressing schemes called **big endian** and **little endian** set the format for sending data to a microcomputer. Big-endian format sends the most significant byte first, while little endian sends the least significant byte first. Figure A shows big-endian byte ordering for a 32-bit word; Figure B shows little-endian byte ordering for a 32-bit word.

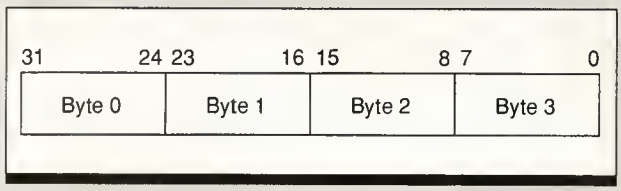


Figure A. Big-endian ordering for a 32-bit word.

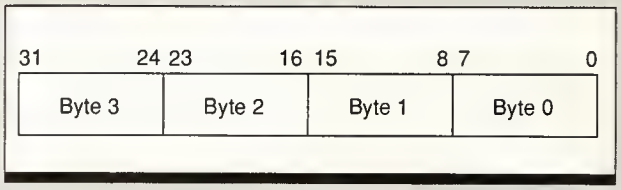


Figure B. Little-endian ordering for a 32-bit word.

A small, high-speed **cache** stores the most frequently used main memory locations. It typically requires only one to two processor cycles to access as compared with 10 to 20 cycles for main memory access. A cache usually can hold 1 Kbyte to 512 Kbytes of data.

The **cache coherency** protocol lets the hardware (or software) ensure that only one logically correct value exists for each program variable. In multiprocessing systems with each CPU containing a local cache, multiple copies of program variables can exist in the system. Each CPU can be attempting to modify and/or access its copy of the program variables simultaneously. The program variable copies could then become inconsistent (with each CPU seeing a different value of a program variable) without some hardware and/or software ensuring that some form of consistency or coherency is enforced.

CISC indicates a complex instruction set computer or computing.

A processor's **condition codes**, or information bits, allow the software programmer (and the hardware) to determine whether the result of a comparison (or other arithmetic operation) was positive, negative, or zero and whether it caused an overflow.

A graphics unit, for a given viewpoint, discards and does not display the nonvisible surfaces of objects in a scene through a process called **hidden-surface elimination.**

A **leaf procedure** will not call any other procedure.

The hardware unit or component called a **memory management unit,** or MMU, translates virtual addresses (those seen by software) into physical addresses (those seen by hardware). The hardware uses the memory-management translation information (page and segment table entries) to translate an address. It then stores the translation information in the TLB. (See TLB.) In addition, the hardware provides program isolation and protection (memory protection) by examining permission data in the translation information.

The **MESI** memory protocol ensures cache coherency between multiple write-back caches. Any given cache entry, depending on how it has been accessed, falls into one of four states: modified (M), exclusive (E), shared (S), or invalid (I).

A **page** is the smallest managed unit of a virtual memory scheme. The system maintains separate virtual-to-physical translation information (and, in some cases, protection information) for each page.

The **Phong-shading** graphics technique helps a graphics unit shade an object. The graphics unit linearly interpolates the normals at the vertices of a polygon along the edges. Then it interpolates the normals at the edges along a scan line. At each pixel along the scan line, the interpolated normal is used in the lighting model to determine the color at that pixel. For example, consider the triangle with four scan lines in Figure C.

Given the normals at vertices A, B, and C, the unit interpolates the normals along edges AB and AC. Then using the interpolated normals of the two edges at horizontal scan line 2, the unit interpolates the normals along scan line 2 between the edges to calculate the shade for each pixel. The unit then repeats the interpolation for scan lines 3 and 4.[3]

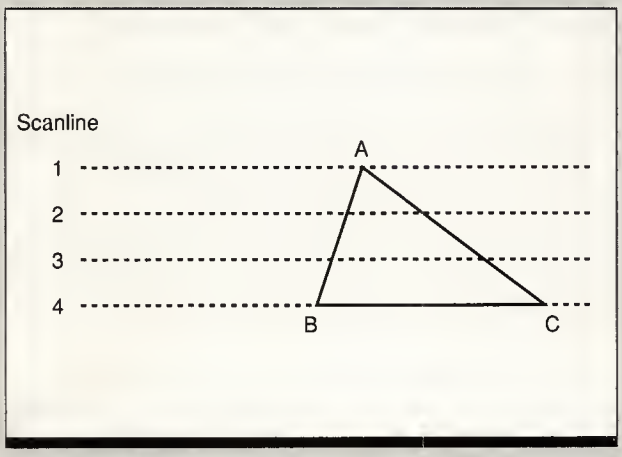


Figure C. A 4-scanline figure.

In a register organization scheme called **register windowing** a group of register banks (each with, for example, 32 registers) is arranged as a circular buffer. During execution, software is only "aware" of a single bank, or window, of registers. Each procedure call, however, results in a new window of registers being transparently allocated to the new procedure, thereby eliminating the need to save registers on each procedure call. Similarly, as each procedure completes and returns, the system readjusts the current window back to the correct window.

RISC indicates a reduced instruction set computer or computing.

A **semaphore** is a hardware/software flag that indicates the status of an activity. Typically, it signals whether or not a shared resource can be accessed. A **semaphore instruction** is a special atomic instruction for accessing the flag.

Smalltalk is a high-level, object-oriented programming language developed by Xerox PARC.

A **spin-on-the-lock** situation occurs when a program is in a loop constantly testing a semaphore to see if access to the related resource is allowed. Here, the semaphore functions like a lock.

An **SPL level** is a Unix interrupt level in a system that supports multiple levels of interrupts. A higher priority interrupt would logically be at a higher SPL level than a lower priority interrupt. Only those incoming interrupts at a higher SPL level than the current one actually cause an interrupt to be acknowledged by the processor. Raising or lowering the SPL level thereby increases or decreases the number of interrupts that the processor will acknowledge. Running at a high SPL level can essentially disable the processor from acknowledging interrupts.

Tagged arithmetic provides a primitive means of checking for consistency in data type thereby supporting the most frequent cases in the Smalltalk and Lisp languages. Many languages, including Smalltalk and Lisp, do not provide data type declarations. Therefore the type (pointer, data) of a program variable cannot be checked until the program is executed. Thus the operand type (and whether they match or not) must be checked before performing arithmetic instructions.

A **test-and-set** instruction typically tests a memory location (flag/semaphore) and updates it according to the flag's value. It is atomic in that after the flag is read and before the flag is updated, the CPU executing the instruction will not allow access to the flag.

A graphics unit transforms a displayed wireframe drawing into a **3D shaded** object by saving only the lines that form the surface of the wireframe's polygons and then filling in (shading) between these lines.

A **TLB,** or translation lookaside buffer, is the memory cache of the most recently used page table entries within the MMU.

A **TLB hit rate** is the cache hit rate achieved in the TLB. It reflects the percentage of memory accesses whose translation information (page table entry) is contained in the TLB.

Virtual memory is the memory space as understood by the software programmer. It allows each software application to "see" a uniform, large address space independent of the number of applications running on a system or the actual size of main memory of that system. The MMU maps, or translates, virtual memory into the actual (physical) memory of the system.

A dynamic, RAM-resident ***z*-buffer** supports graphics processing. It has a one-to-one correspondence with a frame buffer. That is, each pixel in the frame buffer has a corresponding location in the *z*-buffer. Each *z*-buffer location contains the depth (*z*) value (the pixel's distance from the viewer) of the object being displayed at the corresponding pixel in the frame buffer. Before drawing a pixel, the graphics unit compares the *z* value of the object, at that pixel, with the value in the *z*-buffer. The unit updates the pixel only if the object is closer to the viewer than that indicated currently in the frame buffer.

Why architecture instead of implementation? The RISC architectures offer large performance increases over currently available CISC architectures. As a result, almost every computer vendor evaluates the RISC architectures to verify performance claims and to determine which, if any, best fits its applications and strategic directions. Unlike past chip decisions (for example, whether to use an 8086 or an 68000) however, selecting a RISC chip and breaking user software compatibility with current product lines now becomes a major corporate decision. Software compatibility is increasingly important due to the rise of industry-standard application binary interfaces (ABIs). In addition, the number of available user application packages continues to grow. As a result, changes to an architecture become very difficult while changes to an implementation become increasingly easy.

Changing an architecture, in general, implies that changes will have to be made to user application software. Since most computer vendors do not write all of their own applications and because of the enormous number of packages that would have to be updated, the cost of such a change is very high. In some cases additions to an architecture could be made in such a manner that existing user application software is both forward- and backward-compatible. In general, however this is not the case, and the selection of a good architecture is critical.

Changes to an implementation, however, imply that only changes to the hardware and possibly the operating system software will be necessary. Since vendors upgrade both the hardware and operating system on a regular basis (to include the latest chip implementation), the added cost of changes in an implementation remains small in comparison to the user software changes. Any limitations in a given implementation can be (and usually are) reduced or circumvented in the next implementation. Therefore the selection of a RISC based on a given implementation is not as critical.

Overall then, the more important selection criteria in selecting a RISC chip is its architecture, both the current and future implementations as opposed to just the architecture's current implementation.

Architecture evaluation examples. In the process of evaluating RISC architectures, we have seen numerous performance comparisons and architectural "evaluations" based on these comparisons. (Most have been conducted, it seems, by the companies selling one architecture or another.) While these evaluations have been extensive and have pointed out numerous potential shortcomings, most of these evaluations compare specific implementations of the architectures in question and not the architectures themselves. As a result, the shortcomings tend to be characteristic of the implementations and not the architectures. Table 1 lists some of the architectural shortcomings being put forth for each of the architectures.

Table 1.
Claimed Architectural Shortcomings.

Architecture	Deficiency
i860	No cache coherency for internal caches
88000	No dual-cache tags Only supports MESI model of cache coherency
Sparc	Single address/data bus No separate address adder

In each of these cases, the proclaimed architectural shortcoming is, in fact, a feature of the implementation and not a feature of the architecture. The number of external buses, while a major component of the performance of RISC implementations, is not a feature of the architecture. The number, speed, and width of external buses can be (and is) changed from implementation to implementation without affecting the architecture. The support of cache coherency and the exact form of that support is, again, a crucial feature in the implementation of multiprocessor systems but is not a feature of the architecture. Cache coherency can be added, deleted, or changed without affecting the underlying processor architecture.

While many of the analyses being performed may have concentrated on specific implementations as opposed to the underlying architecture, we point out that the architectures are not without shortcomings nor all equal. On the contrary, the architectures, while on the surface quite similar, are quite different when examined in detail.

Overview of architectures

The i860, 88000, and Sparc are labeled and marketed as RISC architectures. They all satisfy the key aspects of RISC design[4] and share some "prominent" RISC characteristics. These shared key characteristics are:

- single-cycle execution (for most instructions),
- simple load/store interface to memory,
- register-based execution,
- simple fixed-format and fixed-length instructions,
- simple addressing modes,
- large register set or register windows, and
- delayed branch instructions.

For some particular target markets, the vendors have also added sets of instructions that are not frequently used in general-purpose computing. For example, the

i860 provides a set of graphics and vector instructions, the 88000 offers an extensive set of bit-field instructions, and the Sparc includes instructions on tagged data. Probably, to some RISC purists/minimalists, the addition of such seemingly extraneous instruction sets disqualifies their classification as RISC architectures. However, in our opinion, the key point of RISC is the design philosophy of simplicity and efficiency. That is, RISC affords an efficient use of hardware resources via judicious simplification of the semantics of a processor's instruction set and encoding of the instruction set. These special instructions do not preclude the three architectures from being classified as RISC architectures.

To avoid a proliferation of memory management architectures, each of the architectures also includes a memory management definition.

Architectural comparison

In examining the architectures of the i860, 88000, and Sparc, we look closely at the following areas:

- miscellaneous instructions,
- branches,
- memory operands and addressing modes,
- registers,
- data types and alignment,
- floating-point units, and
- memory management.

Miscellaneous instructions. In addition to the standard set of RISC instructions, each architecture includes fairly unique (at least for RISC architectures) instructions targeted for specific applications. The special i860 instructions support graphics processing as well as parallel operation of the integer and floating-point units. The graphics processing instructions include an extensive set of both pipelined and nonpipelined instructions, which support z-buffer operations, Phong shading, and pixel arithmetic. These capabilities provide superior support in graphics applications that perform hidden-surface elimination and 3D shading. However, since these instructions use the software-visible floating-point pipeline, their use is limited to libraries and specially coded routines. (We discuss this aspect further later.) For applications outside of the graphics area, these capabilities will not provide any measurable benefits.

The i860 also supports the parallel initiation of the integer and floating-point units via the dual-instruction-mode prefix. Use of this prefix causes the next two instructions to be initiated in parallel (assuming that one is an integer instruction and one is a floating-point instruction). For general-purpose applications, which typically perform few floating-point operations, the addition of such parallelism does not provide any significant benefit. Alternatively, for those applications that perform extensive floating-point operations, such parallelism provides a significant performance improvement. However, since the compiler must generate different code to take advantage of the parallelism (and the current compiler does not), it is unclear whether high-level-language programs will be able to make use of this capability. To the extent that an application's key routines and libraries can be written in assembly language, much of the performance improvement can be achieved.

The unique 88000 instructions are an extensive set of bit-field instructions. They provide the capability to set/clear and extract/insert values into bit fields of variable length and position. (Further discussion appears later.)

The unique Sparc instructions support tagged arithmetic. They provide the capability to tag data and pointers differently so that detection of illegal operations on the data or pointers can be detected. (We discuss this further later.)

Semaphores. The three architectures support some kind of semaphore or atomic test-and-set type of instruction. Semaphore instructions are an increasingly important part of the architecture due to the increase in the number of shared-memory multiprocessing systems being developed. Such systems require semaphores to ensure that the multiple processors of the system modify system data structures in a consistent manner.

The i860 supports a general Lock and Unlock instruction pair, which causes the processor to run all of the instructions between them in an atomic manner with interrupts blocked. (Note that the hardware enforces a limit of 32 such instructions.)

The 88000 supports the XMEM instruction, which loads a memory location, tests it for 0, and if a 0 is detected, stores the specified register contents into the memory location. The load/stores are indivisible on the bus.

The Sparc architecture supports two types of semaphore instructions (though early implementations only support one). The Load-Store Unsigned Byte instruction reads a memory location and then writes that memory location to all 1s in an atomic manner. The Swap instruction causes a memory location to be read and then replaced with the contents of a specified register.

In comparison, it would appear that the i860 Lock/Unlock mechanism provides better support for such things as counting semaphores. However, in fact, the actual number of instructions required to implement such a construct (and therefore the speed to execute it) is approximately the same for all three architectures. Both general mechanisms, the Sparc/88000 and the i860, require multiple instructions to obtain a lock, increment or decrement the semaphore, and then re-

lease the lock. None of the three architectures provides a single-instruction implementation as in the IBM S/370.[5]

Two potential, but small, benefits of the i860 mechanism in an application using the Unix operating system exist. One is its ability to spin on the lock at a low SPL level (interrupt level), and the other is its ability to perform short semaphore or other operations without raising the SPL level at all. In the first case, the Unix kernel requires that the SPL level be raised before attempting to obtain a lock that could also be required at a higher interrupt level. This requirement normally means that software on a processor such as the 88000 or Sparc must raise the SPL level to ensure that it does not get interrupted after obtaining the lock. (If it were interrupted, a deadlock situation could arise.) However, since the i860 Lock/Unlock mechanism blocks interrupts, the SPL level does not have to be raised until the lock has been successfully obtained. In addition, if the work performed on the semaphore or the desired code is short enough (less than 32 instructions), the i860 mechanism allows the software to keep the SPL level the same. In total, however, both of these benefits are small and not of sufficient size to consider further.

Multiply/divide. Of the three architectures, only the 88000 provides both of the basic integer multiply and divide instructions. The i860 architecture supplies a multiply operation via its FMLOW floating-point operation but provides a library routine for division. The Sparc architecture, alternatively, provides a Multiply Step instruction and library routines to implement both multiply and divide operations. The lack of these instructions constrains the i860 and Sparc architectures in measurably increasing multiplication and division performance by using any hardware available in future implementations. As such, i860 and Sparc implementation performance will suffer on applications that require extensive multiplication and division operations unless vendors add the basic multiply and divide instructions to the architecture.

Branches. The three architectures have the concept of a delayed branch. Here the instruction sequentially following the branch executes independently of whether the branch is or is not taken. This feature increases performance of pipeline implementations by reducing the flushing effect of branches on the pipeline. Studies have indicated that this technique is successful in eliminating the branch penalty in 60-70 percent of the cases.[6]

In addition, the three architectures have the ability to essentially annul the execution of the instruction in the delay slot. This provision eliminates the potential increase in code size identified after having to fill the delay slot with a NO-OP instruction. Avoiding this increase reduces the factor by which the RISC code size will increase over a traditional CISC architecture.[7]

None of the three architectures incorporates branch prediction in the instruction set as in the AT&T Crisp[8] architecture. Such software prediction would reduce the branch penalty. However, all of the architectures could adopt any one of the many hardware branch-prediction strategies for a particular implementation.[9] While studies have shown that software branch prediction may be more cost effective to implement, the hardware schemes are not excessively expensive and do provide very good branch prediction.[9]

Additional comparison and looping support. In addition to the usual branch instructions, the i860 architecture provides additional support for those loop operations that terminate with a comparison against 0 via the BLA (branch on loop condition code and add) instruction. This single instruction decrements a counter, compares it to 0, and then branches on that comparison—all in one cycle.

In comparison, the 88000 and Sparc architectures require two instructions (and two cycles) to implement the same functionality. In the 88000 architecture the first instruction decrements the counter. Meanwhile the second instruction compares the result against 0 (creating an intermediate set of condition codes) and executes the branch operation. In the Sparc architecture the first instruction decrements the counter (and sets the condition codes). The second instruction executes the branch operation (based on the condition codes).

However for loops not terminated by a test against 0, all three architectures require a total of three instructions to perform the decrement, comparison, and branch operations. Studies have shown that while loops with a termination of 0 are common, they are not the predominant case.[10] Therefore though the i860 provides better performance in this case, the total performance improvement overall will not be large.

Condition codes. The three architectures support condition codes on which some or all of their branch instructions perform a test.

The i860 provides both the traditional condition-code approach and the loop control instruction just described, which uses a separate condition code. Unlike the Sparc and 88000, however, the i860 arithmetic instructions always set the condition codes. This specification makes the implementation of more complicated pipeline schemes supporting out-of-order execution and multiple-instruction executions per cycle more difficult to implement.

The 88000 architecture departs from the traditional approach of condition codes held in the processor status word. Instead it writes status information resulting from a Compare operation in a general register specified in the Compare instruction. Conditional branch instructions correspondingly test the specified general register to determine whether the branch operation should proceed. Given that no separate condition codes

Five different addressing modes can be synthesized by each architecture.

exist, future implementations of the architecture will more easily employ complicated pipelining schemes supporting out-of-order execution and multiple instruction execution per cycle.

The Sparc architecture allows many instructions to set the condition codes. In addition it provides an explicit Compare instruction and all of its branch instructions test the condition codes. Arithmetic instructions can optionally set the condition codes or leave them unaffected. These provisions will enable future implementations of the architecture to more easily employ the same pipelining schemes as described for the 88000.

While the traditional method has offered separate condition codes, arguments have been put forth against condition codes. They add difficulties to the hardware design and result in an unorthogonal instruction set. The 88000 addressed certain concerns[11] by having the condition-code bits stored in any specified register, as described earlier. This requirement minimized any hardware implementation problems and facilitated the hardware support of parallel integer and floating-point operations. It also effectively eliminated yet another of the few registers that are available to the user. However, given the magnitude of the difficulties associated with using condition codes, any additional hardware that may be required would be small.

Addressing modes. The three architectures share two basic addressing modes for operand access. They are base + offset and base + index. With register 0 returning 0 all the time, five different addressing modes can actually be synthesized. They are:

- *register:* R*x*, where *x* is the register number;
- *register indirect:* (R*x*), where *x* is the register number;
- *register indirect with index:* (R*x*, R*y*), where *x* and *y* are the register numbers;
- *register indirect with immediate offset:* offset(R*x*), where *x* is the register number; and
- *immediate, signed and unsigned.*

In the register indirect with immediate offset mode, the i860, 88000, and Sparc support 16-bit signed offset, 16-bit unsigned offset, and 13-bit signed offset forms, respectively. Given that long immediate offset mode is rarely used, the difference in the length of immediate offset mode is irrelevant. However, support of the signed immediate mode provides some extra flexibility over the unsigned immediate mode.

In the immediate mode, the i860 architecture supports the 16-bit signed immediate form for arithmetic operation and 16-bit unsigned immediate form for logical operation. The 88000 architecture supports the 16-bit unsigned immediate form. The Sparc architecture supports only the 13-bit signed immediate form. Given that long immediate modes are rarely used, the difference in the length of immediate modes is irrelevant. However, the support of the signed immediate mode provides some extra flexibility over the unsigned immediate mode.

It is interesting to note that the above addressing modes are also the five most frequently used addressing modes in CISC machines.[12, 13] In fact, the least frequently used address mode of the five, register indirect with index, has a frequency of only 6 percent.[13]

The 88000 also supports index mode with scaling. This addressing mode simplifies index computation for accessing halfword arrays as well as word arrays. The addressing mode is useful for artificial intelligence languages and scientific computing. However, it will have a low frequency of usage in a general-purpose computing environment. Hence, little performance gain will be seen.

To eliminate the requirement of an additional read port to its register file, the i860 memory store instruction does not support the use of register indirect with index mode. This absence of support introduces asymmetry to the instruction set and hence an exception to the compiler. However, based on a CISC-machine study,[14] less that 4 percent of the second operand and the destination operand in a triadic operation use the address mode. Therefore, we see very little performance impact for the lack of it. For floating-point vector-processing performance, the i860 supports the autoincrement mode for constant stride vector addressing. Since very little floating-point vector processing occurs in general-purpose computing, we again see very little performance impact.

Control-transfer address. All three architectures provide two addressing methods for control-transfer operations, PC-relative and register indirect. For PC-relative conditional transfer, the i860 provides 16-bit and 26-bit offset modes, the 88000 provides 16-bit offset, and the Sparc provides 22-bit offset. The i860 offers a better range for PC-relative transfers. However, based on the previously mentioned CISC-machine study, a 16-bit offset mode sufficiently processes 93 percent of PC-relative branches. A 15-bit offset mode is sufficient for 87 percent of PC-relative branches.[14]

Given the code expansion due to the RISC architecture and the trend in program-size growth, a 16-bit offset mode will probably be good for close to 87

All three architectures provide more registers than their CISC forebears.

percent of all PC-relative branches. Since 15-20 percent of the instructions executed are nonprocedure call-related, PC-relative control-transfers, only 2 percent additional branches are needed to reach the branch target. The penalty of a shorter 16-bit offset mode is insignificant.

For unconditional transfer and procedure call or return, the three architectures provide both register indirect and PC-relative addressing modes.

Registers. The number of application-usable registers becomes a key factor in the performance of RISC processors, given the relative performance penalties associated with accessing variables in cache and/or main memory. This factor and the increasingly sophisticated register-allocation schemes of today's compilers form the primary driving forces behind incorporating a larger set of registers into the architectures of current processors.[15, 16] In this area, all three of the architectures provide substantially more registers than their CISC forebears. However, the registers differ in the way they are used and the number that are available.

The 88000 is the weakest in this area with only thirty-two 32-bit registers for both integer and floating-point operations. Given that each floating-point operand typically takes two registers, the effective number of values that can be contained in the register file is much less than 32. In comparison, the i860 and Sparc with thirty-two 32-bit integer registers and an additional thirty-two 32-bit floating-point registers can hold a substantially larger number of values in the register file. Studies have indicated that this increased number of registers should result in better performance for the i860 and the Sparc.[15]

In addition to the 32 integer registers directly addressable via the instruction set, the Sparc architecture also supports a register-windowing system. This system provides between two and 32 windows of registers arranged as a circular buffer. (For a detailed explanation see the *Sparc Architecture Manual* and Patterson and Sequin.[17, 18])

Proponents of this and similar register-windowing schemes argue that the windowing provides a number of benefits. Among them are:

1) The compiler does not have to save/restore registers across function calls, thereby increasing the speed of the function calls.

2) The compiler does not have to be as complex because it does not have to perform sophisticated register allocation.

3) The windowing system provides a mechanism for providing an increased number of windows in a user software-transparent manner.

Meanwhile, detractors argue that windowing has potential drawbacks:

1) The overflow or underflow of the circular buffer (running out of usable windows) requires that some portion of the windows must be flushed or filled.

2) Context switches now involve the save/restore function of significantly more registers than in the traditional case.

The exact value of a register-windowing scheme (such as that supported by Sparc) in comparison with the use of sophisticated register-allocation techniques (such as those used by the i860 and 88000) has been the subject of several investigations.[16, 17, 19] The studies show that the relative performance of the two options is essentially equal and that the register-windowing scheme provides better performance in some cases.

The relative disadvantages of the register-windowing scheme turn out to be few because the frequency of overflows/underflows and context switches is small in comparison with the frequency of procedure calls. However, not all cases achieved the relative advantages of the register-windowing scheme due to the newer, sophisticated approaches to register allocation that take advantage of program characteristics (such as the high percentage of time spent in leaf procedures).

In addition to these architectural aspects, a major contention of the proponents of register allocation is that the implementation of a register window-based architecture will suffer from having to support the register windows. In particular, they point out that the frequencies of two implementations (one having register windows and one having a typical register file) will not be the same given equal technology because the register windows will require additional logic in the critical path. While this contention has yet to be proven (current Sparc implementations run at frequencies just as fast or faster than the i860 and 88000 frequencies), it could affect certain implementations. However, architectures with register-windowing schemes can support any number of windows including one (same as the register allocation approach) or two (depending on the exact implementation, for example, Sparc requires two). Any negative effects of windowing in such an implementation could be reduced or eliminated as necessary by reducing the window count to a low level. (Though any "old" code, presumably compiled without sophisticated register allocation, would run poorly in such an implementation.)

Byte ordering. The i860 and the 88000 support byte-ordering formats called big endian and little endian. The Sparc supports the big-endian format. The selec-

tion of the byte-ordering method becomes a data-compatibility issue with existing architectures. The Sparc architecture originated at Sun Microsystems Inc., where most of the products are Motorola 680X0-based (big-endian byte order). Big-endian format thus becomes the logical choice. Similarly, the Intel 80X86 line supports the little-endian format, a logical choice therefore for the i860.

The i860 and 88000 support both byte orderings statically. As a result, data can be exchanged with a big-endian machine or a little-endian machine without reversing the bytes or changing the byte numbering. Thus, the i860 and 88000 provide a migration path for data and databases generated from machines of either byte orderings. However, the 88000 ABI specifies the big-endian format (Motorola's 680X0 format) for interfacing to the operating system. Any application running in little-endian byte order must somehow swap the bytes to interface to the operating system. It is not yet clear what byte order the i860 ABI will specify. However, to maintain some sort of data compatibility with the Intel 80486 line, the i860 ABI will probably adopt the little-endian format. Again, any application running in big-endian byte order must somehow swap the bytes to interface to the operating system.

Note also that none of the three architectures provides a complete data-compatibility solution. The majority of the existing machines supports arbitrary byte alignment for data, whereas all three architectures do not. Considering the cost of breaking instruction compatibility (migrating from CISC to RISC), the data incompatibility issue is minor.

Data types. The three architectures supply the usual set of integer data types, namely, byte, unsigned byte, halfword, unsigned halfword, word, and unsigned word.

The three architectures also supply the usual set of ANSI/IEEE floating-point data types, namely, single-precision and double-precision.[20] In addition, the Sparc supports extended-precision floating-point operations, giving it an edge for applications requiring additional precision. While current language standards do not support extended-precision floating-point data, note that as RISC implementations approach mainframe performance the demand for extended-precision floating-point data will increase.

For different target markets, the three architectures support additional data types. The i860 supports 8-bit, 16-bit, and 32-bit pixels to provide high-performance 3D graphics processing. The 88000 supports bit-field data. However, it is limited to data within a word. It has a much narrower range of applications than the Motorola 68020 bit-field instructions that operate across word boundaries. The Sparc supports tagged data. The support of this data type has been shown to provide a 10-25 percent execution-time savings for systems using dynamic data typing, for example, Smalltalk.[21] Since these special data types are really targeted for specific applications, the support of such data types and related operations will not have any performance impact on general-purpose computing.

Floating-point arithmetic. The three architectures support the *ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic*[20] through different levels and mixes of hardware and software emulations. They supply the usual set of floating-point instructions, namely, load/store, integer to floating point, floating point to integer, add, subtract, multiply, and compare.

The Sparc and the 88000 supply division and square-root instructions, whereas the i860 supports the division and square-root functions via reciprocals, a similar approach taken by Cray supercomputers. Here, a Newton-Raphson iterative sequence using the multiply and reciprocal instructions performs a division or square-root operation. As a result, i860 implementations will suffer on those applications that require extensive division and square-root operations. However in general, these operations have low usage frequencies. Measurements taken from an execution of the SPICE circuit simulator on an MOS memory cell circuit show that floating-point arithmetic occurs only 12 percent of the overall time.[22] Out of that 12 percent, division occurs only 9 percent of the time. In other words, the overall usage is 1 percent.

The i860 floating-point architecture supports both scalar and pipelined modes. However, the pipelines are exposed. This means that either software compatibility may have to be broken in the future or a restriction be placed on future implementations.

The i860 also has a set of instructions that can initiate an add/subtract and a multiplication, and control the data paths between the adder and the multiplier pipelines. Vector operations, like multiply and accumulate, can be synthesized (by controlling the data paths accordingly) and be speeded up considerably. However, it is questionable how well a compiler can vectorize and make use of the exposed pipeline. To take full advantage of the vector processing, an application programmer will probably have to make calls to a hand-coded library of vector-processing routines. Again, the i860 vector/pipeline operations are useful for a particular market, and we see little vectorizing/performance for general-purpose use.

Memory management. The three architectures support fairly traditional memory management architectures though each provides additional support in many crucial areas. All three architectures support a full 4-Gbyte virtual address space.[23-25] While this space will be sufficient in the near term, all three will have to deal with larger virtual address spaces in the longer term (a la HP Spectrum[26] and IBM 801 and PC RT[27]). In all three cases, retaining compatibility will be a major architectural challenge.

The i860 supplies a 4-Gbyte physical address spectrum, while the 88000 supports an 8-Gbyte spectrum and the Sparc supports a 64-Gbyte spectrum. The i860 and Sparc share their respective address spaces between the user and the operating system with the exact boundary not being fixed in hardware. Alternatively, the 88000 hardware divides the 8-Gbyte space into 4 Gbytes reserved for the operating system and 4 Gbytes for the user. The ability to directly address spaces of greater than 4 Gbytes will become increasingly important in future systems with multi-gigabyte main memories and with 32-bit, direct-addressed input/output buses. Sparc sufficiently addresses this need with its 64-Gbyte address space, while both the 88000 and i860 restrict space to more traditionally sized physical address spectrums. Fortunately, both the i860 and 88000 have reserved bits in their page table entries. These bits could be used to increase their physical address spectrum in the future.

The 88000 and the i860 support two levels of address translation while the Sparc supports three-level translation. Theoretically, the two-level translation will reduce the time to translate the virtual address into a physical address when the translation cache or the lookaside buffer does not contain the translation information. However, the overall effect is small due to the high TLB hit rates. A detrimental effect of only having two levels of translation, on the other hand, is the overhead (in terms of the number of pages required) encountered to map the large, sparse address spaces of processes in the Unix operating system. The adoption of Unix System V Release 4.0 along with the increased number of logical segments used in applications (shared libraries, mapped files, etc.) makes it increasingly important to reduce the overhead of the page tables associated with each process.

All of the architectures support a 4-Kbyte page size. While larger than many page sizes in traditional CISC architectures, the increased size of applications (as well as the increased size of RISC-executable files) justifies the use of a large page size. Even larger page sizes (more than 4 Kbytes) are good for systems with a large amount of memory and running relatively few large applications (workstations). They are not suitable for systems with a small amount of memory and running numerous small applications. For a system with a fixed amount of memory, for instance 8 Mbytes, a 4-Kbyte page size results in a "pool" of 2,000 pages. An 8-Kbyte page size results in a pool of only 1,000 pages. For applications with a large number of small processes, higher performance will be achieved with systems holding 2,000 pages in the pool rather than 1,000.

Given the small number of TLB entries available in the microprocessor implementations of these architectures, only a small amount of virtual address space can be mapped without incurring TLB miss penalties. If only pages are supported in a memory management architecture, a typical TLB implementation with 64 entries will map only 64 × 4 Kbytes, or 256 Kbytes of memory. While such a mapping size is sufficient for most user applications with their high degree of locality, it is not enough for large applications or the Unix kernel, which exhibit a very low degree of locality. Therefore, support of some larger form of mapping, for example, segments, is required to provide sufficient performance. In addition, such large mappings require large, continuous pieces of physical memory. Many applications such as the Unix kernel really use only a portion of multiple mappings (for text and stack). Therefore, it is important that the mappings not be too large to minimize the wastage of physical memory. (Though some of it can be effectively used by double-mapping this area of physical memory.)

Both the Sparc and 88000 architectures support such a larger mapping. The 88000 supports 4-Mbyte mappings with the option to individually enable or disable 256-Kbyte "chunks" of that mapping. The Sparc, alternatively, supports 256-Kbyte, 16-Mbyte, and 4-Gbyte mappings. The ability of both architectures to effectively map 256-Kbyte pieces of the address space sufficiently addresses the problem of the low locality and at the same time minimizes the wastage of physical memory.

The i860, however, does not support any form of larger mappings. This deficiency will result in a much lower effective TLB hit rate, which could severely impact overall system performance in some applications. Support of some kind of large mapping facility could be added, however, since this feature is typically not visible to the user and is hidden by the kernel (the virtual memory subsystem in Unix V Release 4.0). Also, the most crucial application of the larger mapping appears for the kernel when a change from pages to a larger mapping would be entirely invisible to the user.

The three architectures provide the minimum user/kernel and read/write protections. Sparc, in addition to these minimum permissions, also offers a limited combination of Execute permissions. The addition of Execute permissions provides Sparc with capabilities that will be useful in dealing with such things as dynamic shared libraries.

Overall, the i860, 88000, and Sparc memory management architectures provide essentially equal capabilities with the exception of the lack of large mapping support in the i860. The Sparc architecture offers the most flexibility and possibilities for future growth. But all three architectures will require significant upgrades when virtual address spaces of greater than 4 Gbytes become important.

In summary, examination of the various components of the overall architectures reveals that each have some areas that offer better support than the others and some areas that provide worse support. Table 2 summarizes the assessments of the various components

Table 2.
Relative Architecture Support.

Area	i860	88000	Sparc
General			
Unique instructions	≥	≥	≥
Semaphores	=	=	=
Multiply/divide	≤	=	≤
Branches	≥	=	=
Addressing modes	=	=	=
Registers	=	≤	=
Data types	=	=	=
Floating-point functions	≤	=	=
Memory management	≤	=	≥

of the architectures that we examined, the i860, the 88000, and the Sparc. For each of the components in the table, we indicate whether we found that the architecture was slightly inferior with respect to the others (≤), essentially equal to the others (=), or slightly superior to the others(≥).

The i860 architecture is weaker in the floating-point area because of the software-visible pipelines, in the memory management area because of its lack of support of a large memory mapping, and in the higher math area due to its lack of a full divide instruction. However, the i860 architecture is stronger in the branch area because of its loop control support instruction. The 88000 architecture is weaker in the area of registers because of the smaller number of registers that the architecture supports. The Sparc architecture is weaker in the area of higher math functions due to its lack of support for full multiply and divide instructions. However, the Sparc architecture is stronger in the memory management area because of its more flexible MMU, or memory management unit, and additional page permissions.

Of the relative weaknesses that were identified, they vary in how difficult they would be to change. The lack of a large mapping in the i860 could be remedied by the addition of such a construct to the MMU. Since this construct will most importantly be used by the kernel, its addition could be made entirely user-software transparent. The software visibility of the floating-point pipelines in the i860, alternatively, most likely cannot be addressed without significantly breaking software compatibility. As in the Sparc case, the addition of a full divide instruction could be added fairly easily.

The number of registers supported in the 88000 architecture would be very difficult, if not impossible, to increase because of the lack of extra, unallocated, bits within the instruction encodings. The lack of full multiply and divide instructions in the Sparc architecture could be fairly easily addressed using an available free opcode number. Such a change could provide both forward and backward software compatibility (assuming the old implementations trapped onto the new instruction). However, new code would run at unacceptably slow rates on old implementations.

In addition to their general support of typical architectural features, each architecture will provide particular applications with much better support than the others due to special architectural features.

1) The i860 provides the best graphics support with its pixel instructions and data types.
2) The 88000 offers the best bit-manipulation support.
3) The Sparc provides the best artificial intelligence support with its tagged arithmetic instructions.

From a system implementation point of view, the three architectures support the basic primitives necessary to implement a general-purpose Unix system implementation. While the primitives may be somewhat more primitive that those in traditional CISC architectures, they do provide the basic building blocks upon which a Unix system can be based. In fact, since the building blocks are relatively primitive, they avoid locking in a particular implementation. For example, a CISC context switch instruction gives an implementation the freedom necessary to create a more optimal solution.

In considering all the factors, we find that no one of the three architectures is clearly inferior or clearly superior to the other architectures. A particularly bad or a particularly good implementation of any of these three architectures will more than make up for any architectural differences that have been identified.

References

1. G.A. Blaauw, *Digital System Implementation,* Prentice Hall, Englewood Cliffs, N.J., 1976, p. 2.
2. H.S. Stone, *High Performance Computer Architecture,* Addison-Wesley, Reading, Mass., 1987, pp. 1-20.
3. J.D. Foley and A. van Dam, *Fundamentals of Interactive Computer Graphics,* Addison-Wesley, 1982.
4. W. Stalling, "Reduced Instruction Set Computer Architecture," *Proc. IEEE,* Vol. 76, No. 1, CS Press, Los Alamitos, Calif., Jan. 1988, pp. 38-55.
5. *IBM System/370 Principles of Operation,* 9th ed., International Business Machines Corp., Poughkeepsie, N. Y., Oct. 1981.
6. D.J. Lilja, "Reducing the Branch Penalty in Pipelined Processors," *Computer,* Vol. 21, No. 7, July 1988, pp. 47-55.
7. J.A. DeRosa and H.M. Levy, "An Evaluation of Branch

Architectures," *Proc. 14th Ann. Symp. Computer Architectures*, CS Press, June 1987, pp. 10-16.

8. D.R. Ditzel and H.R. McLellan, "Branch Folding in the CRISP Microprocessor: Reducing Branch Delay to Zero," *Proc. 14th Ann. Symp. Computer Architecture*, CS Press, pp. 2-9.
9. J.K.F. Lee and A.J. Smith, "Branch Prediction Strategies and Branch Target Buffer Design," *Computer*, Jan. 1984, pp. 6-22.
10. M.G.H. Katevenis, "Reduced Instruction Set Computer Architectures," MIT Press, Cambridge, Mass., 1985.
11. J. Hennessey et al., "Hardware/Software Tradeoffs for Increased Performance," *ACM Proc. Symp. Architectural Support for Programming Languages and Operating Systems*, Mar. 1982, pp. 2-11.
12. C.A. Wiecek, "A Case Study of VAX-11 Instruction Set Usage for Compiler Executions," *ACM Proc. Symp. Architectural Support for Programming Languages and Operating System*, Mar. 1982, pp. 177-184.
13. D.W. Clark and H.M. Levy, "Measurement and Analysis of Instruction Use in the VAX-11/780," *ACM/IEEE Proc. 9th Ann. Symp. Computer Architecture*, 1982, pp. 9-17.
14. B.L. Peuto and L.J. Shustek, "An Instruction Timing Model of CPU Performance," *Proc. Fourth Ann. Symp. on Computer Architecture*, Mar. 1977, pp. 165-178.
15. F. Chow and J. Hennessey, "Register Allocation by Priority-based Coloring," *Proc. ACM SIGPlan 84 Symp. Compiler Construction*, June 1984, pp. 222-232.
16. D.W. Wall, "Register Windows vs. Register Allocation," *Proc. ACM SIGPlan 88 Symp. Programming Language Design and Implementation*, June 1988, pp. 67-78.
17. *Sparc Architecture Manual*, Sun Microsystems, Inc., Mountain View, Calif., 1987.
18. D.A. Patterson and C.H. Sequin, "A VLSI RISC," *Computer*, Vol. 15, No. 9, Sept. 1982, pp. 8-21.
19. R.P. Colwell et al., "Computers, Complexity and Controversy," *Computer*, Vol. 18, No. 9, Sept. 1985, pp. 8-19.
20. *ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic*, CS Press, 1985.
21. D.M. Ungar, *The Design and Evaluation of a High Performance Smalltalk System*, MIT Press, Cambridge, Mass., 1986.
22. W. Hollingsworth, H. Sachs, and A.J. Smith, "The Clipper Processor: Instruction Set Architecture and Implementation," *Comm. ACM*, Feb. 1989, pp. 200-219.
23. *The Sparc Reference MMU Architecture, Rev. 1.3*, Sun Microsystems, Inc., Oct. 1988.
24. *i860 Programmer's Reference Manual*, Intel Corp., Santa Clara, Calif., Feb. 1988.
25. *M88000 Architecture Specification*, Motorola Corp., Schaumburg, Ill., 1986.
26. M.J. Mahon et al., "Hewlett-Packard Precision Architecture: The Processor," *Hewlett-Packard J.*, Aug. 1986, pp. 4-22.
27. "The 801 Minicomputer," *IBM J. Research and Development*, Vol. 27, May 1983, pp. 237-246.

Richard S. Piepho

William S. Wu

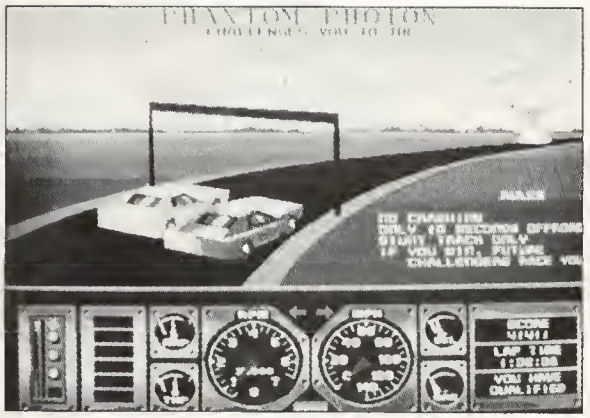
Richard S. Piepho is a member of the technical staff at AT&T Bell Laboratories in Naperville, Illinois. He currently works on the development of future AT&T computer systems. His past work has included performance and architectural analysis of the company's RISC processor, Crisp, as well as the University of California at Berkeley's RISC I processor.

Piepho received a BSEE from Purdue University and an MS in computer science and electrical engineering from the University of California at Berkeley. He is a member of the IEEE Computer Society, Tau Beta Pi, and Eta Kappa Nu.

William S. Wu was the AT&T Bell Laboratories architect for the 32-bit WE32200 microprocessor as well as a member of the design team. His interests include very large scale integration architecture, interconnection networks, and multiprocessor architecture. Wu received a BSEE from the University of Minnesota, an MSEE from Carnegie Mellon University, and a PhD from the University of Michigan. He is a member of the IEEE Computer Society, the ACM, Eta Kappa Nu, and Tau Beta Pi.

Questions concerning this article can be addressed to Richard S. Piepho, AT&T Computer Systems, 1100 E. Warrenville Road, Naperville, IL 60566, or William S. Wu, AT&T Data Systems Group, Crawfords Corner, Holmdel, NJ 07733.

A Fixed-Point DSP for Graphics Engines



Atari's Hard Drivin' video game employs the ADSP-2100 for 3D effects.

High-end graphics processing in scientific laboratories generally requires 32-bit, floating-point machines to provide a greater computational resolution and dynamic range than 16-bit, fixed-point machines can offer. This high resolution minimizes the accumulation of error in the recursive (computationally intensive) transformations that are characteristic of high-end graphics applications. In fact, a floating-point format is often necessary to provide sufficient dynamic range for scaling and zooming operations. However, a low-end graphics engine that uses a 16-bit, fixed-point format is more than adequate for applications such as video games and small computer graphics packages. Here we present an example of this type of application using the 16-bit ADSP-2100 digital signal microprocessor developed by Analog Devices. (See the box on the next page for an explanation of the graphics operations and terminology used in this article.)

In addition to digital signal processing, the ADSP-2100 performs the kind of arithmetic required to process graphics.

Matthew Johnson
Analog Devices

Graphics processing system

Figure 1 on p. 65 is a block diagram of a simple graphics processing system that uses the 2100 processor. An analog-to-digital converter (ADC) takes samples of the joystick positions and furnishes them to the 2100 as input. The processor executes a program that rotates the joystick data as an object that is displayed on an oscilloscope. A digital-to-analog converter (DAC) generates beam-deflection voltages for the oscilloscope from the output data of the graphics processor. An address decoder activates control signals for the converters and maps these devices into the data-memory address space. (A hardware data-memory acknowledge protocol, or DMACK, allows the use of converters that are slower than the processor.)

A four-channel, 8-bit ADC is mapped into the 2100 data-memory space and provides joystick input samples to the 2100 program, which controls the amount of rotation.

The reference object is stored in data memory as a series of (x, y, z) coordinate sets, or vectors. Each vector represents a point or vertex of the object. All vertices are numbered. A manually generated line list stored in data memory describes where (between which points) the oscilloscope is to draw the lines.

0272-1732/89/0800-0063$01.00 © 1989 IEEE

Graphics Operations and Terminology

In graphics processing, geometric and topological information constitute the essence of the image. By contrast, in the more familiar image processing, an image consists of pixels, or the smallest resolvable dots in that type of display. Graphics programs operate on data—and data structures—such as line lists and polygons—rather than pixels. Graphics processors use rotation, projection, and other techniques to *synthesize original* images, whereas image processors *enhance existing* image aspects such as contrast and definition. Graphics applications include operations such as

- geometric modeling and drafting,
- solids and surface modeling,
- hidden-line removal,
- shadow casting,
- texture mapping,
- perspective views,
- image synthesis,
- 3D imagery, and
- animation.

We define a number of terms used in this article.

Data format refers to the size (number of bits in width) and type (unsigned magnitude versus two's complement) of data words in a finite-precision machine. A data format also helps to identify the location and precision of a data word in which *M.N* denotes the number of bits to the left (*M*) and to the right (*N*) of the binary point. The sum $M + N$ defines the word width, that is, a 1.15 data format indicates that one integer bit and 15 fractional bits comprise a 16-bit word.

The *fixed-point* numerical data type represents values that fit within the confines of the full-scale values; special care must be taken to avoid data overflow and underflow. Fixed-point numbers typically use 8- or 16-bit precision and have a fixed resolution with no dynamic range or exponent. The binary point generally does not move, and resolution suffers as values approach the endpoints of the representable range.

The *floating-point* numerical data type consists of a sign, an exponent, and a mantissa. These numbers have great dynamic range and resolution because they have an exponent and a typically large mantissa. This range and resolution assists with overflow and underflow problems. Standard word sizes are 32 bits for single precision and 64 bits for double precision. Adjusting the exponent moves the binary point and maximizes resolution.

Normalization is the process of scaling a set of numbers to comply with some upper limit, usually unity. It also occurs during the process of adjusting results of arithmetic operations so that the destination format complies with the operand format.

When a numerical value exceeds the maximum representable value of the specified data format, the resulting *overflow condition* causes information loss.

A *point vector* equals a set of scalar coordinates such as (x, y, and z) that describe the location of a point in the space defined by the coordinate set.

A *scale factor* equals the set of scale coefficients of the transformation matrix diagonal that individually scales each (x, y, and z) component processed by the transformation matrix. If also refers to the coercion of a set of numbers into another data format by moving the binary point so the hardware can readily process the data.

A *transformation* occurs through multiplication of a point vector with a *matrix* that consists of coefficients corresponding to various types of motion. A *perspective transformation* is a subset of transformation matrix coefficients that individually moves existing vanishing points in from infinity along X, Y, and Z axes. This action actually converges parallel lines to create a more realistic rendering of the object. A *rotational transformation* moves each point vector in a circular fashion through the dimensions of the coordinate space. A *scaling transformation* proportionally moves each point vector through the dimensions of the coordinate space. A *translational transformation* refers to linear movement of point vectors. In a *zooming tranformation*, a single coefficient of the transformation matrix uniformly scales point vectors through all the dimensions of the coordinate space simultaneously.

When a numerical value is less than the minimum representable value of the specified data format, the resulting *underflow condition* causes information loss.

The *unsigned-magnitude*, fixed-point data format ranges from zero to $+2^N - 1$, where N equals the word size in bits. Negative numbers are not representable in this format.

Simple connections between vertices generate an object rendering, or *wireframe model*.

READER-SERVICE CARD

IEEE MICRO

August 1989 issue (card void after February 1990)

Name _____

Title _____

Company _____

Address _____

City _____ State_____ ZIP _____

Country _____ Phone (_____) _____

Please send (Circle those you want)

201 Publications catalog
202 Membership information
203 Student membership information
204 Senior membership information
205 *IEEE Micro* subscription information

Reader Interest
(Add comments on the back)

Readers,
Indicate your interest in articles and departments **by circling the appropriate number** (shown on the last page of articles and departments):

150	159	168	177	186
151	160	169	178	187
152	161	170	179	188
153	162	171	180	189
154	163	172	181	190
155	164	173	182	191
156	165	174	183	
157	166	175	184	
158	167	176	185	

Product Information 1
(Circle the numbers to receive product information)

1	21	41	61	81	101	121	141
2	22	42	62	82	102	122	142
3	23	43	63	83	103	123	143
4	24	44	64	84	104	124	144
5	25	45	65	85	105	125	145
6	26	46	66	86	106	126	146
7	27	47	67	87	107	127	147
8	28	48	68	88	108	128	148
9	29	49	69	89	109	129	149
10	30	50	70	90	110	130	—
11	31	51	71	91	111	131	—
12	32	52	72	92	112	132	192
13	33	53	73	93	113	133	193
14	34	54	74	94	114	134	194
15	35	55	75	95	115	135	195
16	36	56	76	96	116	136	196
17	37	57	77	97	117	137	197
18	38	58	78	98	118	138	198
19	39	59	79	99	119	139	199
20	40	60	80	100	120	140	200

IEEE MICRO

August 1989 issue (card void after February 1990)

Name _____

Title _____

Company _____

Address _____

City _____ State_____ ZIP _____

Country _____ Phone (_____) _____

Please send (Circle those you want)

201 Publications catalog
202 Membership information
203 Student membership information
204 Senior membership information
205 *IEEE Micro* subscription information

Reader Interest
(Add comments on the back)

Readers,
Indicate your interest in articles and departments **by circling the appropriate number** (shown on the last page of articles and departments):

150	159	168	177	186
151	160	169	178	187
152	161	170	179	188
153	162	171	180	189
154	163	172	181	190
155	164	173	182	191
156	165	174	183	
157	166	175	184	
158	167	176	185	

Product Information 2
(Circle the numbers to receive product information)

1	21	41	61	81	101	121	141
2	22	42	62	82	102	122	142
3	23	43	63	83	103	123	143
4	24	44	64	84	104	124	144
5	25	45	65	85	105	125	145
6	26	46	66	86	106	126	146
7	27	47	67	87	107	127	147
8	28	48	68	88	108	128	148
9	29	49	69	89	109	129	149
10	30	50	70	90	110	130	—
11	31	51	71	91	111	131	—
12	32	52	72	92	112	132	192
13	33	53	73	93	113	133	193
14	34	54	74	94	114	134	194
15	35	55	75	95	115	135	195
16	36	56	76	96	116	136	196
17	37	57	77	97	117	137	197
18	38	58	78	98	118	138	198
19	39	59	79	99	119	139	199
20	40	60	80	100	120	140	200

SUBSCRIPTION CARD

SUBSCRIBE TO IEEE MICRO

All the facts about today's chips and systems

❑**YES**, sign me up!

If you are a member of the Computer Society or any other IEEE society, pay the member rate of only **$18** for a year's subscription (six issues).

Society: _____ IEEE membership no.: _____

Society members: Subscriptions are annualized. For orders submitted March through August, pay half the full-year rate ($9) for three bimonthly issues.

_____
Full Signature *Date*

_____
Name

_____
Street

_____
City

_____
State/Country *ZIP/Postal Code*

❑**YES**, sign me up!

If you are a member of ACM, ACS, BCS, IEE (UK), IEEE (but not a member of an IEEE society), IECEJ, IPSJ, NSPE, SCS, or other qualified professional technical society, pay the sister-society rate of only **$33** for a year's subscription (six issues).

Organization: _____ Membership no.: _____

❑ Payment enclosed

❑ Charge to ❑ Visa ❑ MasterCard ❑ American Express

Charge-card number

_____

Expiration date

__ __ __ __
Month *Year*

Prices valid through 12/31/89
8/89 Micro

Charge orders also taken by phone:
(714) 821-8380 8 a.m. to 5 p.m. Pacific
Circulation Dept.
10662 Los Vaqueros Cir.
Los Alamitos, CA 90720-2578

Editorial comments

I liked: _____

_____

_____

_____

_____

I disliked: _____

_____

_____

I would like to see: _____

_____

_____

_____

For reader-service inquiries, see other side.

PLACE POSTAGE HERE

PO Box is for reader-service cards only.

IEEE Micro
Reader Service Inquiries
PO Box 16508
North Hollywood, CA 91615-6508

Editorial comments

I liked: _____

_____

_____

_____

_____

I disliked: _____

_____

_____

I would like to see: _____

_____

_____

_____

For reader-service inquiries, see other side.

PLACE POSTAGE HERE

PO Box is for reader-service cards only.

IEEE Micro
Reader Service Inquiries
PO Box 16508
North Hollywood, CA 91615-6508
USA

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 38 LOS ALAMITOS, CA

POSTAGE WILL BE PAID BY ADDRESSEE

IEEE COMPUTER SOCIETY
Circulation Dept.
10662 Los Vaqueros Cir.
Los Alamitos, CA 90720-9804
USA

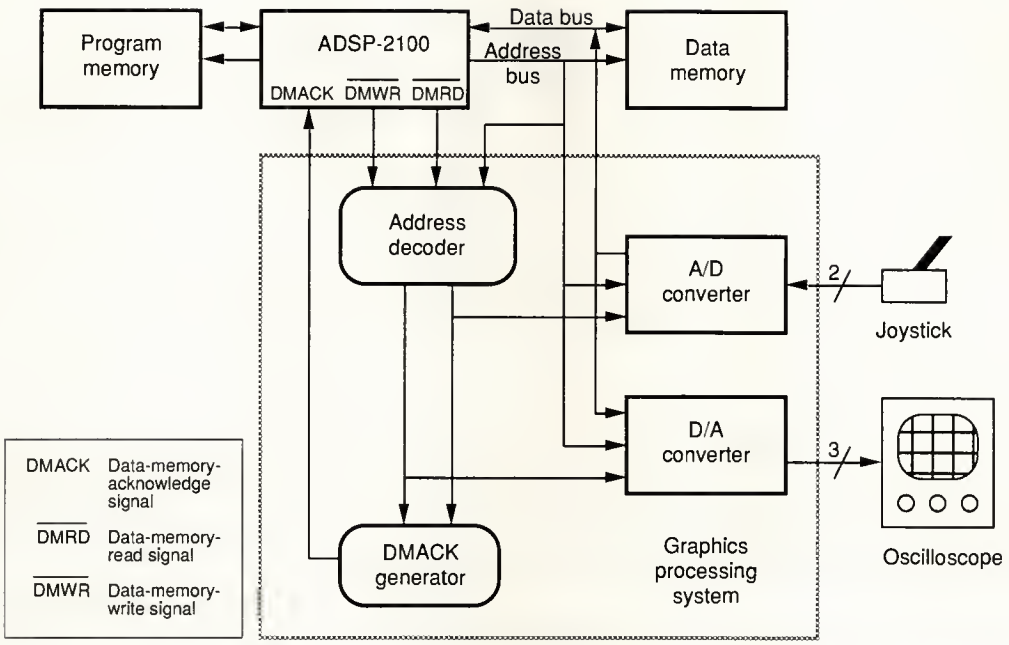


Figure 1. Graphics system block diagram.

The wireframe drawing of the object employs the Bresenham line-segment drawing algorithm.[1] The line list tells the algorithm where the line should be drawn. The 2100 program moves the oscilloscope beam along the path between the two endpoints of the line to perform the actual line drawing. Each line is drawn one pixel at a time, until the entire object has been completed.

Rotation modes. The 2100 graphics software example provides for rotating the object in four different modes. The object can either rotate in each of three dimensions sequentially or in all of them simultaneously. Rotation in these first two modes is automatic and continuous, requiring no joystick control. The object can also rotate in the direction indicated by the joystick.

The joystick position can be sampled and processed in two ways: averaged or filtered. Each of these procedures produces a different effect on the motion of the displayed object (see display driver section). A push button allows the user to switch from one rotation mode to the next.

In the two joystick-controlled modes, each new set of joystick-input samples starts the process of rotating the displayed object. The 2100 program generates a rotational transform from the joystick data to calculate the next position of the object. In the two nonjoystick modes, software automatically generates the rotational transform. In either case, matrix multiplication of the reference object data by the rotational transform calculates the new position of the reference object, point by point. The 2100 program mathematically projects the rotated object from the 3D spatial-coordinate system onto a 2D screen-coordinate system for display. This process is similar to casting the shadow of the object.

Coordinate systems

Scenes that are 3D use a 4D transform space—just as 2D scenes use a 3D transform space—because the (x, y) and (x, y, z) coordinates of 2D and 3D vectors need an additional scale factor. This factor is generally referred to as W. In 2D notation, the point $P(x, y)$ is represented as $P(Wx, Wy, W)$, with the scale factor $W \neq 0$. The coordinates for the point $P(X, Y, W)$ are then $x = X \div W$ and $y = Y \div W$. The scale factor W preserves vector scaling through any transformation. In 3D notation, the point $P(x, y, z)$ is represented as $P(Wx, Wy, Wz, W)$, and the coordinates are recovered similarly.

Scaling factor. Scaling vectors on a point-by-point basis allows equal resolution of coarse- and fine-grained features. For example, one display of a database may show a space shuttle from a distance of 50 meters, while a second view of the same database may detail the 0.25-inch-diameter, 20-pitch-thread, bolt positions inside the pod-bay-door control assembly of the same shuttle.

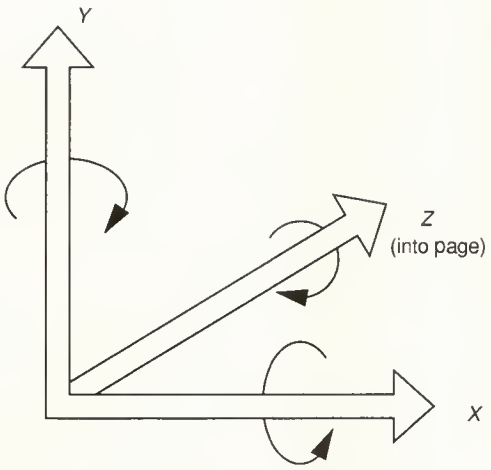


Figure 2. The left-handed coordinate system.

Large values of W allow fine-grained coordinates (that would otherwise be too small to handle) to appear in a fixed-point data format with the same resolution as coarse-grained coordinates. The latter coordinates necessarily have smaller values of W. In essence, the exponent W associates with each fixed-point coordinate set as would the embedded exponent afforded by complete floating-point hardware.

For the sake of simplicity, we set $W = 1$ so that the 3D point $P(x, y, z)$ is represented as $P(X, Y, Z, 1)$, in which $x = X$, and so forth. All points are represented as row vectors with normal scaling and (x, y, z) components:

$$P(X, Y, Z, 1) \equiv [x\ y\ z\ 1] \quad (1)$$

We used the left-handed coordinate system shown in Figure 2 because this system displays larger Z values so that they appear to be further from the viewer than smaller z values. This convention is more intuitive than the familiar right-handed system in which the Z axis comes out of the page. Positive rotations for the left-handed system are always clockwise when one views the origin from a positive axis.

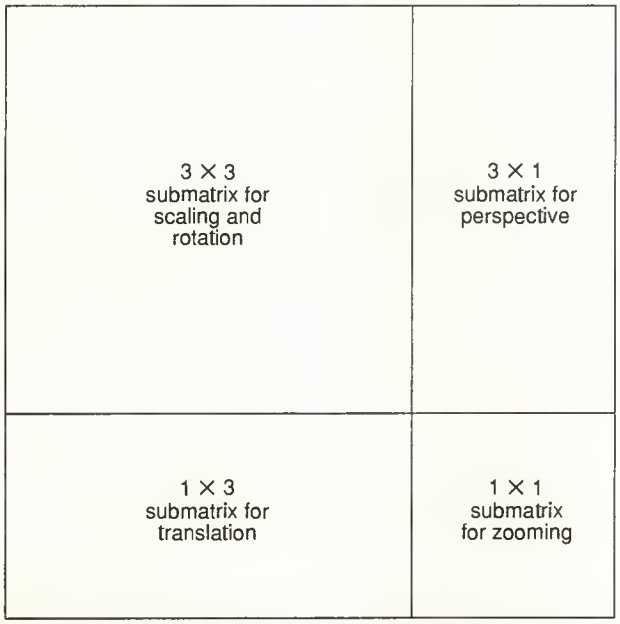


Figure 3. Components of the 4 × 4 transformation matrix.

Transformation matrix. Individual transformations can be concatenated by matrix multiplication to form a single, complex transform. The complex transform has the same total effect on the object as each simple transform applied sequentially, that is, the superposition of linear systems. Thus, multiple operations can be performed simultaneously (rather than sequentially), which saves valuable processor time.

A 4D transformation matrix generally comprises various submatrixes that correspond to different operations (see Figure 3). Rotational operators comprise a 3 × 3 submatrix justified to the upper-left corner of the 4 × 4 matrix. Translation operators constitute a 1 × 3 submatrix in the lower left corner. Perspective operators constitute a 3 × 1 submatrix in the upper right corner. Zooming uses only a 1 × 1 element in the lower right corner.

The conventional geometric operations that can be performed on 3D coordinates (in a 4D space) are rotation, translation, and scaling, as shown in Figure 4. Cx and Sy, for example, denote the trigonometric functions $\cos(x)$ and $\sin(y)$, in which x and y are the angles of rotation. Similarly, this rule can be applied to the rest of the terms. (We disregard perspective transformations and zooming for the moment.)

Computational reductions

One can simplify the transformation matrix to reduce computational requirements. This process also reduces the capabilities of the graphic display system, but the trade-off holds little significance for this application, as we further show in the Performance section.

Any number of rotation, translation, and scaling matrixes can be multiplied together before being applied to the object. The result is always a single 4 × 4 matrix **M** of the form shown in Figure 5.

The upper-left 3 × 3 submatrix **R** gives the aggregate rotation and scaling of all the premultiplied matrixes. The lower-left 1 × 3 submatrix **T** gives the aggregate translation. (Lowercase letters are components.) A reduction in the amount of numerical processing to evaluate the overall transform is obtained by the simplification:

$$[x'\ y'\ z'] = [x\ y\ z] \cdot \mathbf{R} + \mathbf{T} \quad (2)$$

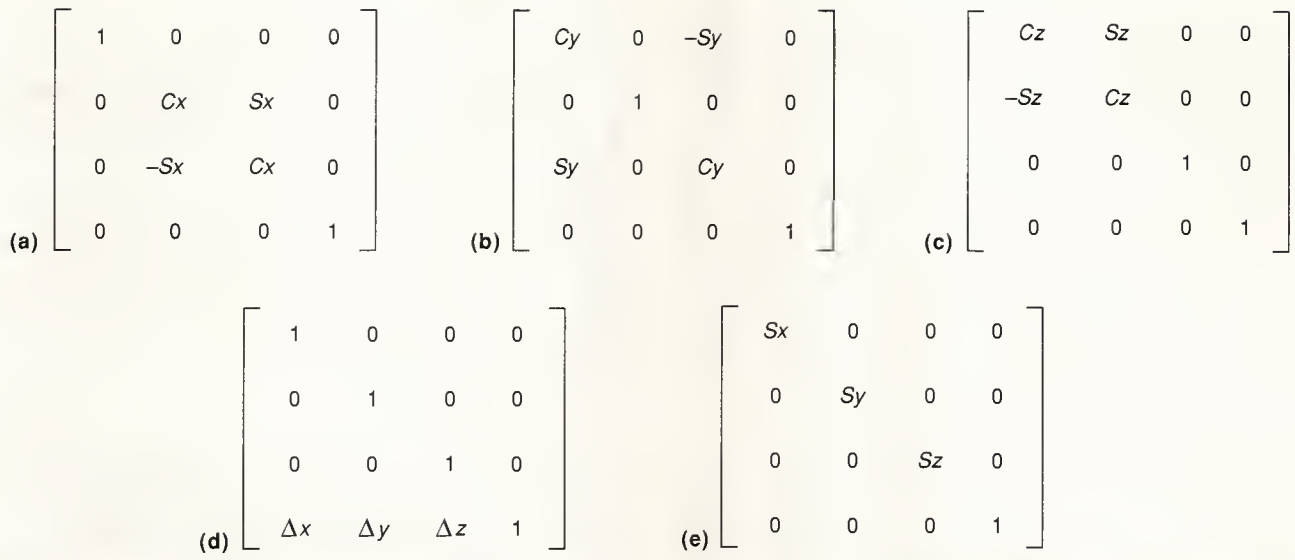


Figure 4. Geometric operations: rotation about the *X* axis (a); rotation about the *Y* axis (b); rotation about the *Z* axis (c); translation by (Δ*x*, Δ*y*, Δ*z*) (d); and scaling by (*Sx*, *Sy*, *Sz*) (e).

rather than by implementing the full $1 \times 4 \cdot 4 \times 4$ multiplication directly in which

$$[x' \; y' \; z' \; 1] = [x \; y \; z \; 1] \cdot \mathbf{M} \qquad (3)$$

The 3×3 matrix structure provides a much simpler and faster implementation because only nine multiplications and six additions are needed to transform each vector, as opposed to 16 multiplications and 12 additions for the 4×4 matrix structure. This process represents a savings of 56 percent in multiplications alone!

The 3×3 matrix structure preserves both rotation and translation, although the zooming and perspective functions are lost. Applications needing zooming must either preserve the 4×4 transform structure and sustain an increased computational load or use the 3×3 structure and apply any zooming operations as a postprocess to the rotational transform. The choice between these options depends on the number of vectors and the nature of throughput requirements.

Because we have no great dynamics in this example application ($W = 1$ for all points), the loss of the zooming function doesn't matter.

Perspective. Perspective transformations introduce realism by use of one or more vanishing points. Without perspective, parallel lines converge at a point located at infinity.

Vanishing points are imaginary points that are set at some finite distance from the object along a major axis. They move the convergence point of parallel lines in from infinity and introduce foreshortening. Foreground objects appear larger and background objects appear smaller. This process, of course, creates an illusion of realism (see Figure 6).

$$\mathbf{M} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$$

Figure 5. Combined rotation, translation, and scaling matrix.

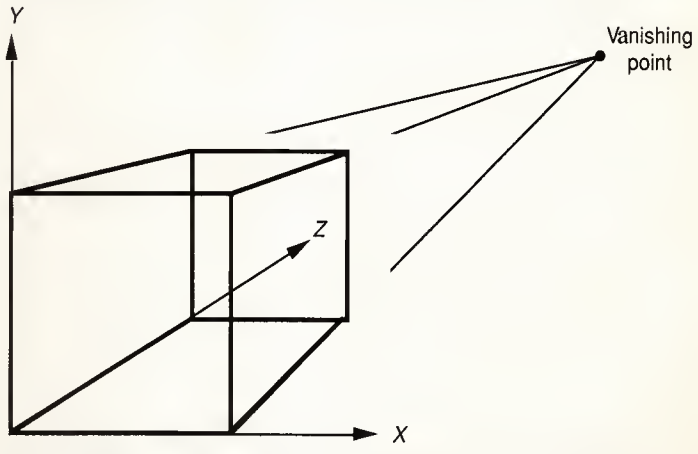


Figure 6. Perspective projection using one vanishing point.

Nonzero elements in the 3 × 1 perspective submatrix migrate the vanishing point associated with the corresponding axis in from infinity. The simplification to the more efficient 3 × 3 matrix structure loses perspective transformations because in this process all perspective transformation elements are assumed to be zero.

We used a simple parallel-projection technique in this example that does not need perspective projection. Therefore, the loss of perspective transformations is not important. If perspective transformations are needed, then the transformation matrix size must increase to the 4 × 4 structure previously described. The problem with this increase in size is that the efficiency of the whole computational process suffers.

Forfeiting perspective and zooming transformations to improve efficiency is a subjective decision. The final visual result determines the correctness of the cost/performance trade-off decision. The criteria for making these decisions consist of aesthetic and performance considerations. If more complex (and realistic) visual effects are needed, using zooming and perspective transformations is the best choice.

The combined rotational transformation matrix **R** used in the example is shown in Figure 7. One should apply any translation (matrix **T**) after calculating **R** with the simple addition we have shown in Equation 2. This example, however, does not demonstrate translation itself.

$$\mathbf{R} = \begin{bmatrix} CyCz & CySz & -Sy \\ SxSyCz - CxSz & SxSySz + CxCz & SxCy \\ CxSyCz + SxSz & CxSySz - SxCz & CxCy \end{bmatrix}$$

Figure 7. Concatenated rotational matrix.

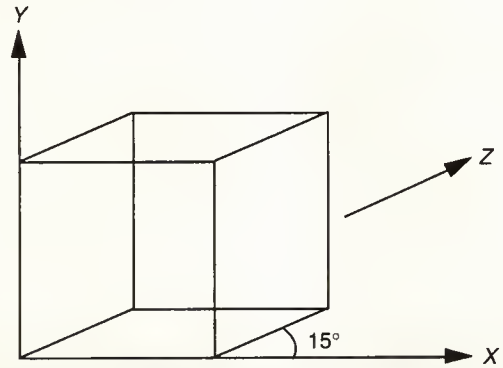


Figure 8. Parallel projection of a cube.

Projection techniques

Once the scene has been transformed in three dimensions, it must be projected onto a 2D screen for viewing. As we mentioned, this operation is like casting a shadow. The sun's rays project a 3D object onto a 2D sidewalk.

We use a simple parallel-projection scheme (see Figure 8) to generate the display data because of its relative simplicity and effective realism. The 2D screen-coordinate (x_s, y_s) display data is derived from the transformed 3D coordinates (x, y, z) using simple trigonometry:

$$x_s = x + z\cos(15°) \quad (4)$$
$$y_s = y + z\sin(15°) \quad (5)$$

Lines parallel to the Z axis appear to make a 15° angle with the X axis (as projected on the screen). This process occurs due to the angle that the projection screen normal and viewpoint (which share a common direction) make with the X-Y plane. Using different angles changes the appearance of the projection.

Note that if the full 4 × 4 matrix structure is utilized, both the transformation and projection operations can combine into a single matrix. We distinguish between these two operations for simplicity and illustration. Foley and van Dam[1] discuss this issue more fully.

Data format

Hardware multipliers don't know the difference between 101.0101_2 and 1010.101_2 (binary; base 2); the placement of the binary point is purely arbitrary as far as the hardware is concerned. However, the 2100 multiplier is optimized for the 1.15 format for the reasons that follow.

The 2100 multiplies two 16-bit numbers and produces a 32-bit product. This result consists of a 16-bit most significant product (MSP), followed by a 16-bit least significant product (LSP). This 32-bit result is then shifted to the left one place and a zero is added for the least significant bit of the LSP. This process has the effect of automatically renormalizing products. The destination format is similar to the input-operand format. No binary-point migration results as long as both input operands are in the 1.15 format. Two multiplicands in that format produce a 32-bit product in the 2.30 format that is shifted one place to the left to produce the 1.31 format. The product is then rounded (or truncated) to the most significant 16-bit half of the 32-bit product, which yields the original 1.15 format. The input format is reproduced at the output. Automatic normalization works only with the 1.15 format. Other formats manifest binary-point migration. Hence, the programmer normalizes all data to the 1.15 format prior to processing. (See Figure 9.)

Note that it is the 16-bit MSP that contains the result in the 1.15 format, although product summing occurs in the full 40-bit resolution of the accumulator before the result is rounded (or truncated) to the MSP.

Normalization, headroom, and scaling

Two input-data operations must occur before the program can run without data overflows. The programmer must normalize all input data to the largest integer value and then adjust all normalized data by scaling from the 16.0 integer format to the 1.15 fractional format.

Normalization. Normalization is the division of a set of numbers by their largest member. After the largest number in the set is normalized to unity, the rest of the numbers are guaranteed to be less than, or equal to, the number one. Data normalization guarantees that products become smaller after multiplication instead of larger. This procedure prevents data overflows.

The programmer scales normalized data to the 1.15 format for automatic renormalization by shifting to the left. Multiplying the normalized data with the 16-bit, two's-complement, positive full-scale value ($7FFF_{16}$ = 32767) scales the data to the 1.15 format ($_{16}$ denotes base 16, hexadecimal numbers).

In the source data of this example, the largest component of any point vector is 21, so normalization would entail the division of all vector components by 21. However, normalization to unity yields a few numbers that are still large enough to cause intermediate results to overflow during the transformation process due to addition operations. Therefore, we increase the normalization factor to guarantee that such overflows are eliminated. By trial and error, we determine that a normalization factor of 30 is sufficient.

Normalization of the source data therefore entails division by 30. For example, the normalized value of 21 is calculated 21 ÷ 30 = 0.7. The normalized data is then multiplied by positive full scale to produce the source data used in the transformation process: for example, 0.7 (32767) = 5999_{16}.

Headroom. The finite precision of the processor's numerical format and the selection of a data normalization factor (resolution) play crucial roles in the successful development of any numerical processing application. Too much resolution in the data (a small normalization constant) results in less headroom (allowance for overflow of intermediate results) within the fixed word size. On the other hand, too little resolution (a large normalization constant) distorts the data. The key to success is to balance the normalization of data with the word size to maintain sufficient headroom and resolution throughout the process.

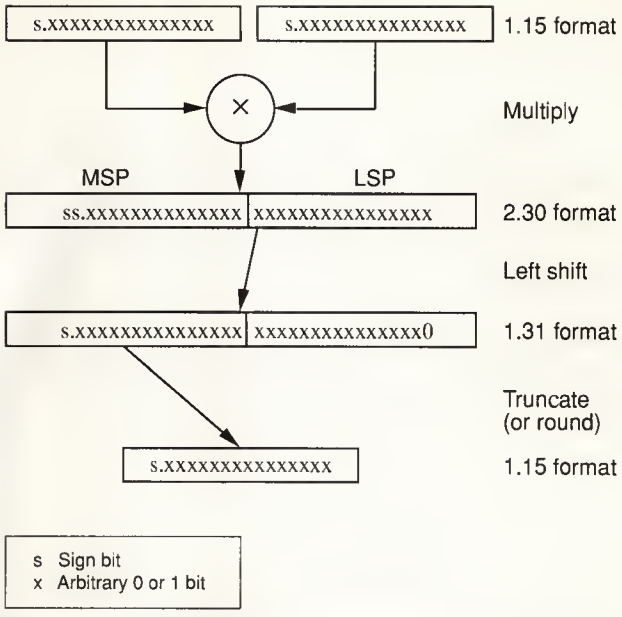


Figure 9. The 1.15 data format.

The two photographs shown in Figure 10 demonstrate the problems that arise when too little headroom is provided. Figure 10a shows a slight overflow of the screen-coordinate system that results in points wrapping around the screen edges. This wraparound process is due to insufficient normalization scaling (not enough headroom), which produces arithmetic overflows.

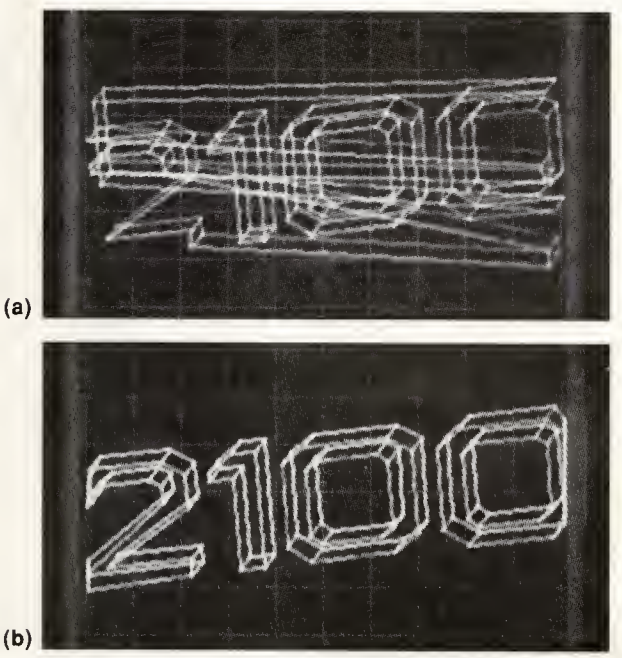


Figure 10. Screen displays of overflows without (a) and with (b) saturation logic.

The photograph in Figure 10b illustrates the use of saturating arithmetic, which is an optional mode of operation on the 2100 ALU and multiplier and accumulator (MAC). The MAC automatically saturates, or sets to full scale, any overflows. Figure 10a illustrates that without saturation logic such wrapped points produce lines that must cross the screen to make their connections. Figure 10b shows how points that would otherwise wrap around the screen are constrained to the edge (clipped). Using saturation arithmetic appreciably reduces the severity of overflow distortion.

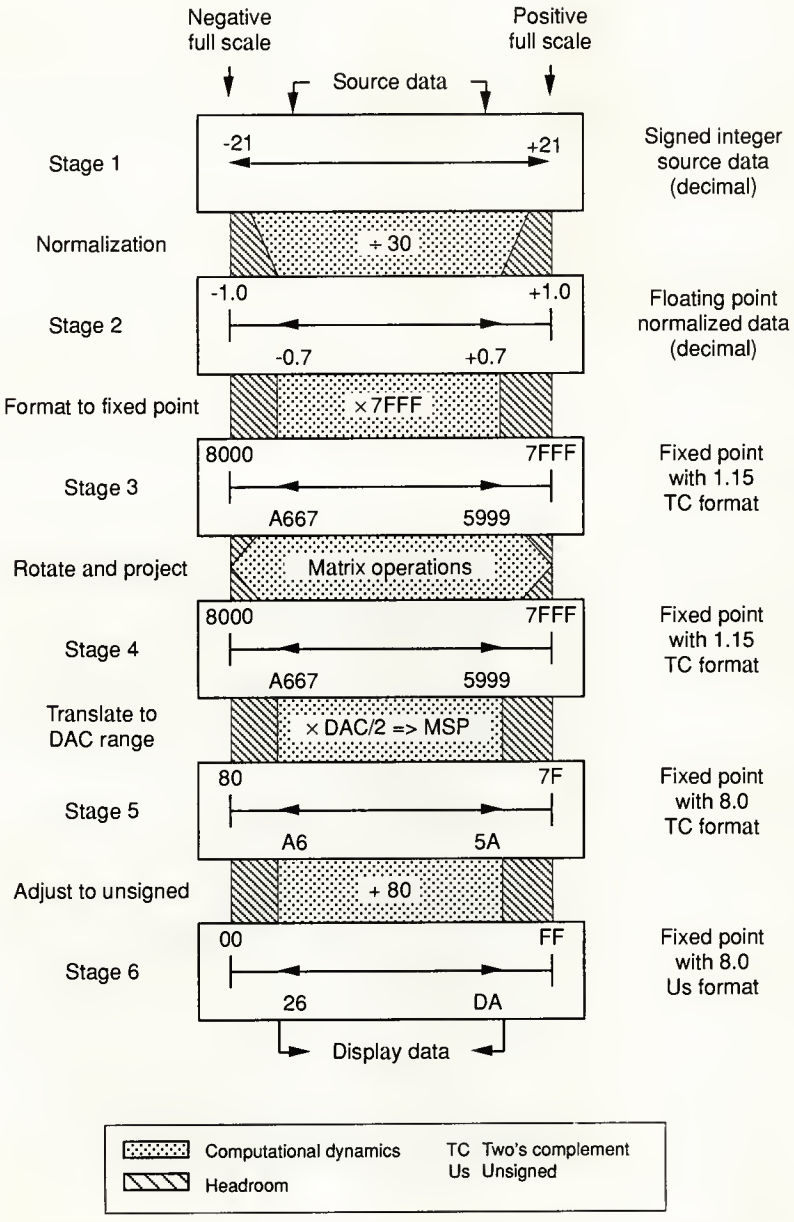


Figure 11. Data-format transition summary.

Scaling. The normalization and scaling of input data are necessary to preserve data integrity through the transformation and projection processes. Before the screen can display the data, however, the output data must be further scaled to adapt to the display driver, which supports yet another data format.

A simple example of a vector-graphic display terminal is the oscilloscope. The hardware we used employs a straight, binary-coded (not two's-complement), four-channel 8-bit DAC (an AD7226) to drive the x and y beam-deflection inputs of an oscilloscope (see Figure 1). The 8-bit DAC provides a screen resolution of 256 × 256 pixels upon which to display the rotating object.

The origin of the 3D coordinate system of the source data is located in the center of the object with points (vertices of the object) that assume ± two's-complement values (corresponding to the format of the 2100) in three dimensions. All 2D display data must therefore be converted to the unsigned-magnitude, 8-bit binary format used by the DAC prior to display. This is done by multiplying each screen coordinate (x_s, y_s) by the DAC's half-scale value (80_{16}) and then adding an offset of half-scale value to shift the center of the object to the DAC's half-scale point.

In our example, the maximum integer value of source coordinates is 21, which—when normalized and converted to the 1.15 format—becomes 5999_{16}. Assuming that the worst case gain through rotation and projection is unity, the maximum display value is 5999_{16}. Prior to being written to the DAC, this value is multiplied by the DAC's half-scale value, 80_{16}, which translates the normalized value to a corresponding voltage of the DAC's output range. The left-shifted resultant product is $(5999 \times 0080 = 0059\ 9900)_{16}$, which rounds to $5A_{16}$, a worst case screen-coordinate value.

Adding 80_{16} to $5A_{16}$ yields DA_{16}. This addition simply moves the object to the center of the screen and has no scaling effect. The final value written to the DAC for display is DA_{16}. Note that the ratio of the worst case screen-coordinate value to the positive full-scale DAC value $(5A:80)_{16}$ is the same as that of the original source coordinate to the normalization factor (21:30).

Figure 11 uses several number-line analogies to summarize the data format transitions and dynamics during operations. It also shows the available headroom for each of the six stages just de-

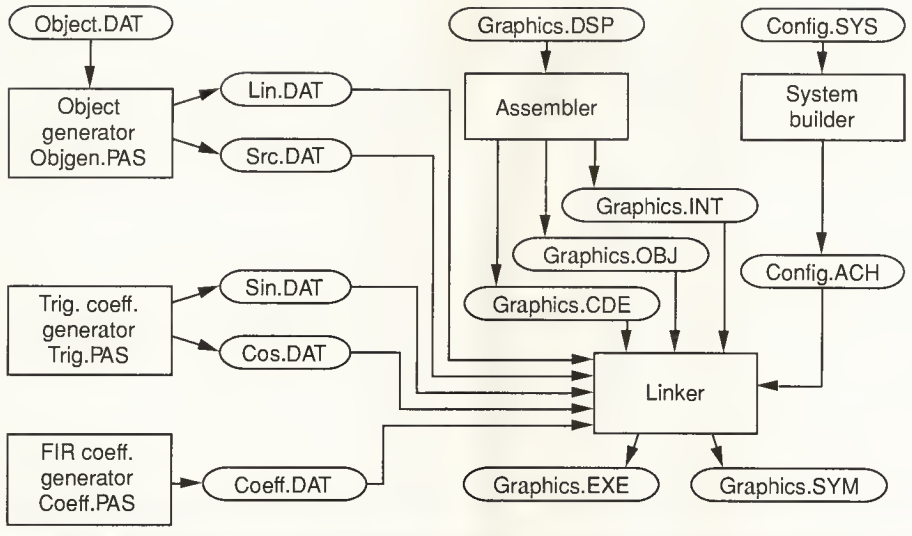


Figure 12. Program and file flowchart.

scribed. In summary, these stages contain the following steps.

1) The programmer edits the actual signed-integer source data consisting of manually quantized (x,y,z) coordinates of the object into a data file (see flowchart in Figure 12).

2) A Pascal program normalizes the quantized floating-point data.

3) The same Pascal program formats the normalized data to produce a hexadecimal fixed-point data file. This data is ultimately loaded into the allocated area of data RAM on the target system by the 2100 assembler initialization directive, INIT, in the main program.

4) After the 2100 has performed rotation and projection transformations, the same limits and headroom are present as in the previous stage. However, during the processing between these two stages, the computational dynamics of the operations in Figure 11 use the headroom to avoid data overflow.

5) The data has been multiplied by the half-scale DAC value (the MSP of the MAC contains the result) to translate the two's-complement data range to a corresponding full-scale range for the DAC. Remember that the two's-complement format provides for only half of the actual range that the unsigned format does.

6) The last step is to compensate the data for the unsigned format of the DAC by adding the half-scale DAC value to all data. This operation moves the two's-complement, negative, full-scale value to zero, zero to mid-scale, and positive full-scale to positive full-scale. The resulting data is what actually defines the vertices of the 2D object between which the line-segment drawing routine draws lines (see the display driver section).

Programs and files

This section describes the programs and files used in the example graphics application. The flowchart in Figure 12 illustrates the various operations and how they interrelate. Ovals represent data files, while rectangles indicate operations or programs. The brief descriptions in this section give general explanations of each file. *The ADSP-2100 Applications Handbook, Vol. 2,* provides complete listings of programs.

Developmental software. The ADSP-2100 Development System aids software design and facilitates program debugging. This software consists of six modules: system builder, assembler, linker, simulator, PROM (programmable read-only memory) splitter, and C compiler. The system builder and the assembler create object code from the graphics source program and system-architecture file. The linker combines this object code with the data files generated by Pascal programs to create executable code. The last three modules are self descriptive.

Object generation. A Pascal program (Objgen.PAS) translates the textual information it receives from Object.DAT representing vector coordinates, connectivity, scaling, and the number of vectors. This program produces hexadecimal versions of the source-vector coordinates (fully normalized and formatted) and the line list (Src.DAT and Lin.DAT data files). These files are used as resources in the main program. Directives load the data into the arrays that were allocated by the assembler.

A 33-percent processor utilization leaves ample time for extras.

Trigonometric coefficient generation. Another Pascal program in Figure 12 (Trig.PAS) generates two files (Sin.DAT and Cos.DAT) that contain 256 uniformly spaced samples of the sine and cosine functions. These functions correspond to the 256 possible positions (the ADC uses 8-bit quantization) that the joystick may assume. These hexadecimal data files are fully normalized and formatted. The lookup tables used during the generation of the transformation matrix are initialized with data from these files. Note that zero is positioned in the middle of the arrays to correspond with a zero rotation at the center joystick position. Directives in the main program load the data in these files into executable code during the linking process.

FIR filter coefficients. A finite-impulse-response program (Coeff.PAS) generates underdamped filter coefficients. These coefficients are used in the filtered joystick display mode (described later) to introduce greater realism by use of inertia. We experimentally derived the response of the filter by plotting various exponentially damped sine waves as a function of its simulated ringing and settling time characteristics. We subjectively considered the best response to be the impulse response of the desired filter. Quantizing this response into 128 samples produced the FIR coefficients. The actual settling time of the filter is the number of taps divided by the frame rate (see the Performance section), or $128 \div 90 \approx 1.5$ seconds. Coefficients were normalized to produce a unity gain filter and then converted to a 1.15 format. The hexadecimal values were stored in a data file (Coeff.DAT) for loading during the linking process.

System configuration. System configuration is mandatory for all 2100 applications. The target-system configuration is specified through an architecture file (Graphics.SYS) as input to the system builder. The memory and peripheral mapping defined in this architecture file must correspond to the target-system memory configuration and peripheral address decoding. This file defines RAM and ROM segments and their locations. Device interfaces are also declared using the PORT directive. Note that in the Config.SYS file data, memory can be interleaved with peripheral devices as long as contiguous data arrays remain smaller than the allocation block size. The system builder produces an output file (Config.ACH) in the format required by the linker.

Main source program. The assembler processes the source code and allocates variable storage. It also produces graphics files (Graphics.INT, Graphics.OBJ, and Graphics.CDE) used by the linker. The linker initializes the various RAM arrays with the data files. It produces an executable image (Graphics.EXE) and a symbol table (Graphics.SYM) that can either be downloaded to a RAM-based system or simulated. Burning ROMs requires the additional formatting step of the PROM splitter.

Only 950 lines of source code are used here (which approximate 2,000 lines of executable code). As indicated by performance benchmarks (see the Performance section), much of the code consists of loop constructs. The main loop takes about 94,000 instruction cycles to complete one iteration (including all iterations of inner loops). Various allocation and initialization steps are performed at startup before the program enters the main loop. This loop consists of building a new transform, applying the transform to the object, and projecting and displaying the object. The loop continuously repeats. A manual button on the target board generates an interrupt to the 2100, whose service routine sequences between the four display modes.

The display routine uses an interesting technique: an indexed, indirect Jump. A Jump table (see Figure A in the accompanying box) consists of different Jump Label instructions. An index into the Jump table is created and added to its base address. Then, an indirect Jump into the table is performed. The index determines which Jump instruction in the table executes.

Display driver

The display driver draws the wireframe object on the screen. The line list describes the endpoints for drawing lines that form the polygons that comprise the object. Zeros in the line list indicate to the line-drawing routine that it should jump to the next point without drawing a line, as in the start of a new polygon. This procedure is equivalent to that produced by plotter Penup commands. Nonzero numbers in the line list tell the line-drawing routine to draw a line from the last point to the next point (as in a Pendown command), which is identified by the number itself. An A – 1 value (8000_{16}) in the line list indicates that no more points remain and the drawing is complete.

Four display modes are demonstrated in our graphics application:

- automatic, sequential rotation about the X, Y, and Z axes;
- automatic, simultaneous rotation about all three axes;
- averaged joystick control in the X and Y axes; and
- filtered joystick control in the X and Y axes.

Indirect Jump Table

The code in Figure A is part of the display routine. The ALU internal feedback register AF stores the index value. In this case, the index determines in which of eight possible octants the new point is located relative to the last one (see Figure B). The signs (positive or negative) of the Δx and Δy values select a quadrant. The difference in magnitude between the values ($|\Delta x| - |\Delta y|$) selects one of the two octants within that quadrant. The index picks out the Jump instruction to the correct octant routine to draw the line to the new point.

```
                AF=PASS 0;              {init for indirect jump offset}
                AX1=DM(newx);           {compute delta x)
                AY1=DM(oldx);
                AR=AX1-AY1;
                DM(dx)=AR;
                AX1=4;                  {set mask for bit 2)
                IF GT AF=AX1 OR AF;     {set bit 2 if delta x is positive}
                AX1=DM(newy);           {compute delta y}
                AY1=DM(oldy);
                AR=AX1-AY1;
                DM(dy)=AR;
                AX1=2;                  {set mask for bit 1}
                IF GT AF=AX1 OR AF;     {set bit 1 if delta y is positive)
                AX1=DM(dy);             {compute |dx|-|dy|}
                AR=ABS AX1;
                AY1=AR;
                AX1=DM(dx);
                AR=ABS AX1;
                AX1=AR;
                AR=AX1-AY1;
                AX1=1;                  {set mask for bit 0)
                IF GT AF=AX1 OR AF;     {set bit 0 if delta x is greater)
                AX1=^jump_table;        {fetch jump table base address}
                AR=AX1+AF;              {add computed offset}
                I4=AR;                  {move result to address generator}
                JUMP (I4);              {do the indirect jump)
jump_table:     JUMP octant6;
                JUMP octant5;
                JUMP octant3;
                JUMP octant4;
                JUMP octant7;
                JUMP octant8;
                JUMP octant2;
                JUMP octant1;
```

Figure A. Jump table code segment.

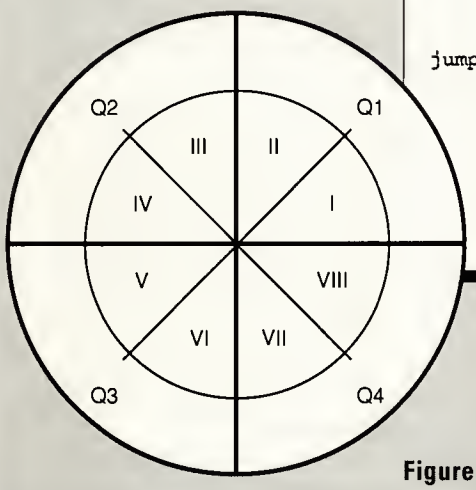


Figure B. Quadrants and octants.

A switch connected to an interrupt input of the 2100 advances the display from one mode to the next.

The first two modes rotate the object about 1.5° per frame, which corresponds to the resolution of the trigonometric coefficient tables previously described (360° revolution ÷ 256 entries/revolution). The averaged joystick mode sums 128 samples and then shifts the result down by 7 bits to produce an average reading for each direction (X axis and Y axis). Averaging reduces the potentiometer jitter associated with the joystick wiper action. The filtered mode applies an FIR filter to both X- and Y-axis readings. Two 128-tap delay lines track historic samples of x and y. These samples are convolved with the FIR coefficients of an exponentially underdamped sine wave.

As mentioned, the actual lines are drawn pixel by pixel using an optimized Bresenham's algorithm (see references and bibliography) for quickly generating line segments between endpoints. Bresenham's algorithm is particularly attractive for this hardware implementation because it requires no division or multiplication, only simple integer arithmetic. Depending on which octant the new point occupies,[2] either the X or Y axis (whichever has the faster rate of change) is incre-

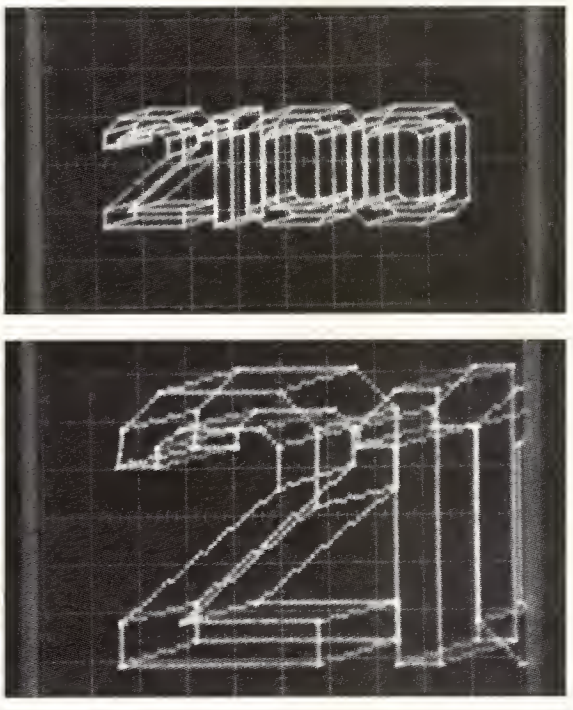


Figure 13. Screen displays of a typical object image (a) and the same image in detail (b).

mented or decremented (depending on the direction) one pixel at a time. At the same time, the other axis is conditionally incremented or decremented. An error term that is tracked with each iteration determines whether a conditional increment or decrement is to occur. Typical object images are shown in Figure 13.

Moves to new points (Penup moves) on the display screen occur with the beam turned off, which is accomplished by writing FF_{16} to the Z-axis DAC. The output of the Z-axis DAC connects to the Z-axis input (or beam intensity) control, which is found on the back of most oscilloscopes. After the DACs are updated with the (x, y) coordinates of each new pixel, a macro turns on the beam for about ten cycles to make the pixel visible and turns it off again. The Z-axis modulation eliminates extraneous display artifacts such as retracing and DAC transitions.

The background register set of the 2100 comprises a complete set of duplicate data registers. The system uses this set during the Bresenham algorithm because other operations (matrix generation, transformation, and projection) have many constants already in memory. A complete set of background registers that can be instantaneously activated makes the time-consuming process of context switching (Push Data, Process New Context, Pop Data) obsolete.

Table 1. Performance benchmarks.

Program-phase functions	Execution duration (cycles)
Generate or gather new position data (modes)	
Autoxyz	26
Autorotate	20
Averaged	2,848
Filtered	330
Generate a new transform	133
Apply the transform	1,927
Project and scale scene to 2D	1,595
Draw scene	89,695

Performance

The object in this application consists of 112 3D vectors and 170 line segments. Its major program loop consists of the functions shown in Table 1. Execution times are shown in terms of the cycles necessary for each execution. Because joystick input samples are either averaged or filtered over 128 items, some modes require less time than others.

The entire main-program loop repeats almost 90 times a second, which is three times faster than necessary for a perception of continuous rotation. In other words, the 2100 can handle an application three times as complex as this example and still convey the illusion of smooth rotation! In fact, the 90 frames/second display rate already includes pleasant performance enhancements—such as the joystick filtering and averaging modes—that are really unnecessary. A 33-percent processor utilization leaves ample processing power for extras such as hidden-line removal, shading and texture mapping, and shadow casting.

The key number in the benchmarks is the 1,927 cycles required for the entire transformation subroutine. This figure measures the number of cycles from the subroutine call to its return and includes all the subroutine-setup overhead instructions. A way to put this benchmark in perspective is to normalize it by the number of transforms that are actually performed: 112 1×3 vectors each multiplied by a 3×3 transformation matrix produces a transformation rate of $1{,}927 \div 112 = 17.21$ cycles/transform. (Although the loop is only nine instructions long, the iterations require some overhead.) Within each 17-odd cycle transform, the following operations are performed:

- fetching the instructions (the cache RAM is used after the first iteration),
- fetching nine coefficients and three vector components,

• performing nine 16-bit multiply/accumulate operations,
• storing three results, and
• maintaining RAM pointers to both the transform and data arrays on each cycle.

The number of cycles in the transformation subroutine equals

[(4 inner-loop instructions × 3 columns) + 5 outer-loop instructions] × 112 vectors + 23 overhead instructions = 1,927 cycles. (6)

If the transformation matrix size increases from 3 × 3 to 4 × 4, the number of cycles would be

[(5 inner-loop instructions × 4 columns) + 5 outer-loop instructions] × 112 vectors + 23 overhead instructions = 2,823 cycles. (7)

This size increase would use 46 percent more cycles. However, its impact in the overall context is relatively insignificant. Tallying these benchmarks for the 3 × 3 structure, we have about 93,373 cycles per frame (using an automatic display mode). This figure corresponds to an 85.7-frame rate using an 8-megahertz processor. A similar tally for the 4 × 4 structure gives about 94,313 cycles, which corresponds to an 84.8-frame rate. (This tally allows for the increased transform time and an estimated increase for the transform build function.) The benchmarks for the new 12.5-MHz ADSP-2100A processor can be derived in a similar fashion.

Using the 4 × 4 structure (that includes the translation, zooming, and perspective operations) would cost a mere 1-percent decrease in the frame rate.

A more significant factor affecting overall performance is the beam-dwell time, or the amount of time the beam dwells at each point. The beam dwell saturates the screen phosphor of the oscilloscope at each pixel position just long enough to leave a nice, bright trace. The value for the beam dwell used in the benchmark measures 10 cycles per pixel. Because the vast majority of the 2100's time is spent drawing lines, variations in the beam-dwell time produce large changes in the overall frame rate. In fact, cutting the beam-dwell time in half increases the frame rate of the 8-MHz processor from 85 to 106 and decreases processor utilization from 35 to 28 percent—while still producing an acceptable display.

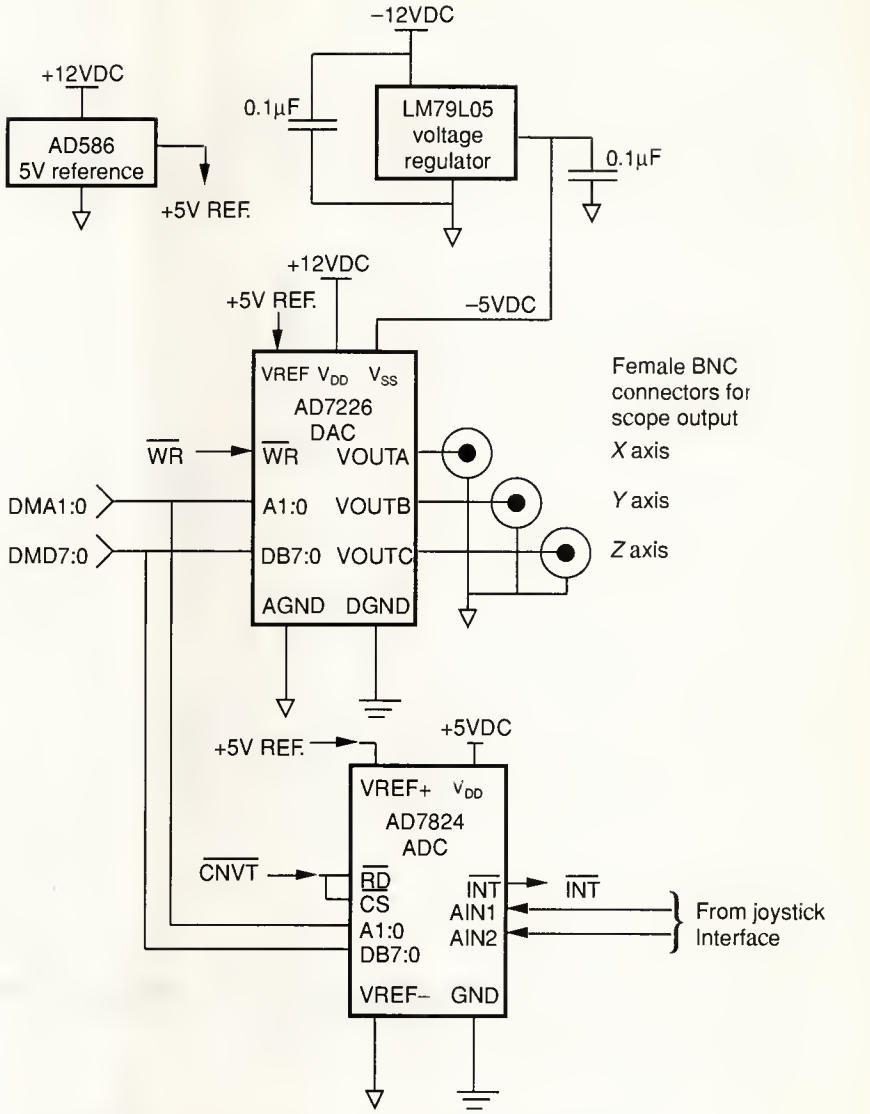


Figure 14. ADC and DAC connections.

Schematics

Figures 14 through 16 detail the schematics for the graphics processor. Figure 14 shows the ADC and DAC connections. The AD7824 is a four-channel, 8-bit ADC

with a 2.4-μs conversion time equal to 20 cycles of the 8-MHz 2100. An AD7226 four-channel, 8-bit DAC drives the scope inputs. The I/O selecter ($\overline{\text{IOSEL}}$) signal in Figure 15 is a predecoded bank select into which both the ADC and the DAC are mapped. Reads from the $\overline{\text{IOSEL}}$ memory region come from the ADC, whereas writes to the same region proceed to the DAC.

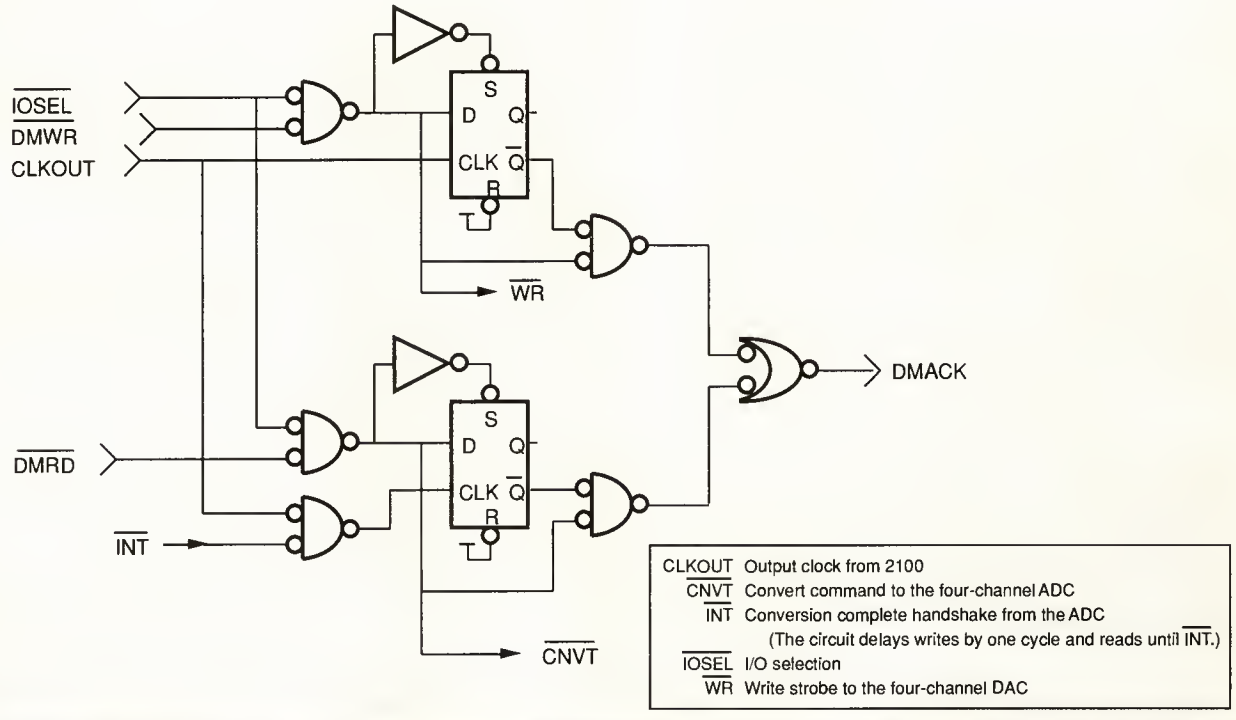


Figure 15. Read/write decoder and DMACK logic.

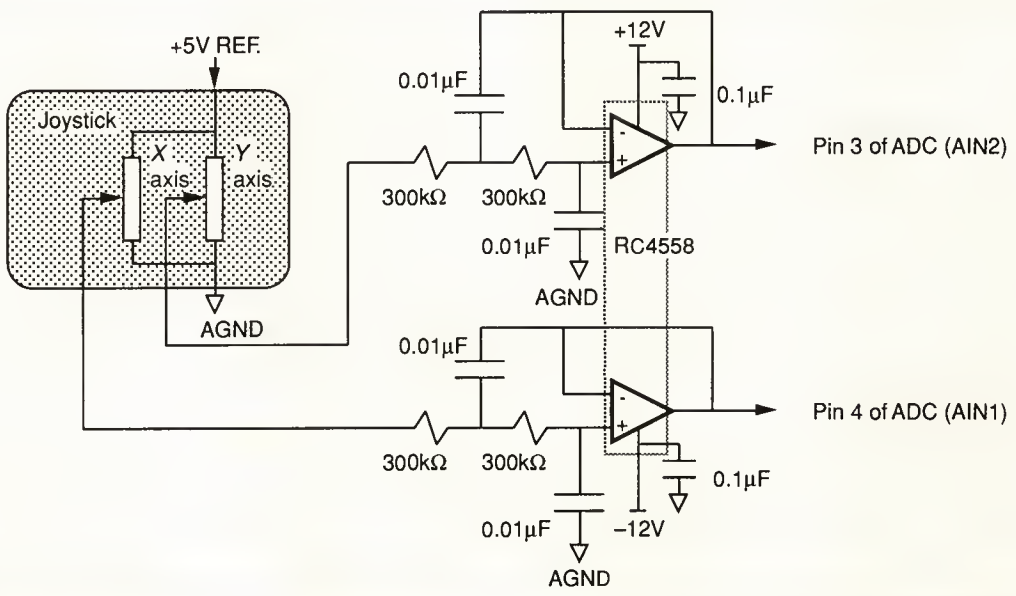


Figure 16. Joystick interface.

The circuit shown in Figure 15 provides read/write decoding and DMACK signal generation. The address decoder processes control signals from the 2100 to provide the Write Strobe ($\overline{WR}$) and Convert Command ($\overline{CNVT}$) control signals for the DAC and ADC, respectively. The Conversion Complete ($\overline{INT}$) output of the ADC, which becomes active low when the conversion is complete, produces the DMACK for the 2100. DMACK generates wait states during ADC and DAC conversions by delaying the 2100 for an appropriate amount of time.

The assertion of $\overline{IOSEL}$ and the Data Memory Read ($\overline{DMRD}$) starts the A/D conversion by issuing the $\overline{CNVT}$ signal to the ADC. The ADC converts within 2.4 μs, during which time DMACK holds the 2100 in a slow peripheral-read mode. Wait states (NOPs) are executed until the conversion is complete. DAC writes also hold off DMACK for one extra cycle to expand the write pulse width of the 2100 to meet the longer requirement of the AD7226's write strobe.

Figure 16 depicts the joystick interface circuit. The RC4558 dual operational amplifier buffers the joystick *X* and *Y* inputs to the ADC. The amplifier also processes some of the joystick potentiometer noise with low-pass filters to stabilize the display.

The ADSP-2100 can serve as the basis for a complete, hardware-oriented application for performing graphics operations on a 3D database. The application presented in this article performs normalization and formatting to avoid overflow and preserve data formats through the transformation operation. It uses data structures that facilitate object rendering through the Bresenham line-segment drawing algorithm. We have derived a 3×3 rotation matrix for this application. We have also described the means for implementing translation, scaling, perspective, and zooming.

Benchmarks show that a 3D object can be rotated smoothly in a real-time display on an oscilloscope. Miscellaneous support software illustrates the basic techniques of generating source data and coefficients and introducing them to the program.

The 2100 proves to be more than capable of handling basic graphics-oriented applications. In fact, the complete application presented in this article uses less than one third of the available processing power of the slowest 8-MHz 2100. An application with three times the complexity could be implemented on the 2100 while maintaining the 30-Hz frame rate needed for smooth display. Note that the latest version of the 2100, the ADSP-2100A, runs at 12.5 MHz.

References

1. J.D. Foley and A. van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Co., Reading, Mass., 1983.
2. *The ADSP-2100 Applications Handbook, Vol. 2*, Analog Devices, Inc., Norwood, Mass., 1988.

Bibliography

Newman, William M., and Robert F. Sproull, *Principals of Interactive Computer Graphics*, McGraw-Hill, Hightstown, N.J., 1983.

Proc. 87 Siggraph Conf., Vol. 21, No. 4, Maureen C. Stone, ed., ACM, New York, 1987.

Matthew Johnson is a senior applications engineer with Analog Devices' DSP Division. Previously, he worked as a senior test engineer at Teradyne in Boston. He is currently developing Huffman image compounding code on the ADSP-2100 and 2101/2 for use in facsimile machines and related applications.

Johnson received his BSEE degree from Worcester Polytechnic Institute. He is a member of Siggraph, the National Computer Graphics Association, the Boston Computer Society, and the IEEE.

Questions about this article can be addressed to the author at Analog Devices, Digital Signal Processing Division, One Technology Way, Norwood, MA 02062

Department

Micro World

Hubert Kirrmann
Asea Brown Boveri Research Center
CRBC.1
CH-5405 Baden, Switzerland

The Smaky story: The Swiss personal computer

Since the potential for personal computing was discovered, each country has tried to get its share of the profits involved. Beside sparing currencies, market participation keeps talents in the region and tightens a country's grip on its technology.

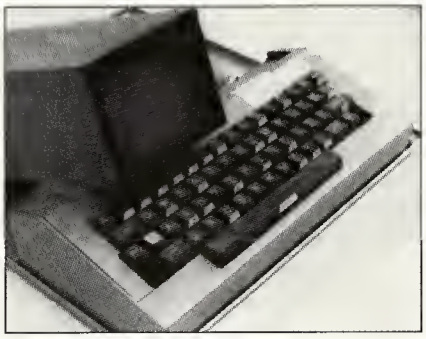
At first glance, making a personal computer does not seem very difficult. Selling it is another matter.

To start with, a successful personal computer must emerge at the same time as a well-tuned manufacturing infrastructure, a rich software base, and an organized distribution network. This combination means huge costs, huge manpower commitments, and high risks. So, many companies turned to the government to lend a helping hand.

One big market that is relatively well shielded is education. A single government contract can create a base market of several thousand machines. The students will learn about the machine and keep on using it afterwards, at least so it is hoped. France, which traditionally strongly supports its national industry, encouraged the development of the Goupil for classroom use. England did the same for its Acorn computer. A corollary is, of course, that no country accepts computers developed by a neighbor country. To top it all off, these machines are tailored to a particular home market and their local strength becomes a problem for export. So, most countries that tried their own way of competing ended up building PC-compatible products.

Switzerland with its 6 million inhabitants has the highest computer density in Europe. But its educational market is small and its protectionism is weak. Nevertheless, Switzerland managed to build a small personal computer industry of its own, which lives on two ingredients: quality and a handful of enthusiastic supporters.

The Swiss micro started in the USA back in 1973, while Jean-Daniel Nicoud was on sabbatical leave at Digital Equipment Corp. in Maynard, Massachusetts. There, he developed the Portable Computing System. PCS had an 8080 microprocessor, 8 Kbytes of RAM, a 5-inch display, and a dual 8-inch floppy drive. Digital Equipment did not want to support a non-PDP-11 processor and discontinued the project.



The Smakey No. 2 (1975): A briefcase-size personal computer with dictaphone cassette on the right corner of the keyboard.

Back in Switzerland at the Federal School of Technology in Lausanne (EPFL) where he still teaches today, Nicoud redesigned the PCS. He replaced the bulky floppies with cassettes, so the whole PC could fit into the case of the keyboard, hence its name: Smaky for SMArt KeYboard. Several versions were built, all with the aim of being so cost-effective that every student could afford to own one. Initially, programs were cross-developed on a Nova; but soon, editors and assemblers appeared on the Smaky itself.

The Smaky 4 quickly became very popular among the students of the LAMI (Nicoud's LAboratoire de MIcroinformatique) and outside EPFL. The fans organized themselves into the Micro Club. In the evening, schoolchildren took possession of the machines of the students for programming games or databases. Many of the future programmers of the Smakys emerged from this group. The CALM assembly language (Common Assembly Language for Microprocessors) was designed to respond to the necessity of using the same syntax for different microprocessors. CALM is being adopted as a standard by the IEC.

In 1977 Bobst, a manufacturer of graphics equipment, asked for a portable computer that a traveling reporter could use to send a paper over telephone lines. Nicoud redesigned the Smaky with two cassettes, a 7-inch screen, and a battery. To reduce the size, he placed the CRT below the keyboard and arranged for its image to be reflected by a parabolic mirror on the suitcase cover. The Scrib, as it was named, was probably the first laptop computer, and Nicoud missed no opportunity to carry its 12 kg (26 lbs., 7.28 oz.) to conferences and demonstrate how he communicated with his lab over the next-available telephone line. A thousand Scribs were built; some are still in use. This collaboration oriented the future Smaky activities toward all computer activities related to text editing and desktop publishing. Juerg Nivergelt at the Swiss Federal Institute of Technology developed its XSO window system on it.

0272-1732/89/0800-0078$01.00 © 1989 IEEE

The 1978 Smaky 6 was a self-contained PC with a Z-80 microprocessor, a 64-Kbyte RAM, a graphics screen, and floppies. It went into production just after the Apple II emerged in the States. These machines were interconnected by an original local area network called Cobus. Cobus allowed them to communicate and access a disk server implemented on an Eclipse S130—long before Appletalk existed. Nicoud placed special emphasis on interfacing external devices like Winchesters or music synthesizers through the parallel bus, MuBus. This explains why the sides of the keyboard case consisted mainly of connectors. The students could, for instance, drive external logic modules to control an elevator. About 500 Smaky 6s were built; many are still in use today.

In 1979 Niklaus Wirth of Zurich (who later designed Modula 2) reported on the Alto he had seen at Xerox. Wirth asked if LAMI could build him a mouse, and Andre Guignard and Rene Sommer developed an optical mouse with a ball. They exported most of them—to the United States, since Xerox was not willing to distribute its own mechanical mouse. This development gave origin to the mouse activity of Logitech, which has since sold 2 million of them all over the world.

The Smaky 8 initially included both a 68000 and a Z-80 processor. The Z-80 ran the existing software and served as the network driver. A faster (800 kilobits/s) and cheaper network called Swan (Single Wire Area Network) interconnected the Smakys to disk servers and to a Ricoh laser printer. The Smaky 8, with its 256-Kbyte RAM, mouse, window screen, and floppies, was an early sister of Apple's Lisa and had the same problem, cost. But the students liked it. It offered for the first time the flavor of desktop publishing—in 1981.

In 1984 Nicoud designed the Smaky 100 as a low-cost Smaky 8 suited for mass production and came out with a machine similar to the Macintosh I. From now on the keyboard and the computer were separate, so one could use two floppy or Winchester drives. The lab's Rene Beuchat developed a new low-cost network, Znet, with a handful of standard ICs to connect machines without drives, since the original Burroughs chips could not be obtained anymore. An enthusiast of the first hour, Daniel Roux, who is the chief Smaky software writer today, designed the text, image, page, music, and font editors. This fact explains the uniformity of the user interface. Roger Hersch developed the graphics interface, Philippe Schweizer and Patrick Faeh wrote the compilers. Beat Brunner wrote the multitasking operating system when he was 18 years old—he had joined the Micro Club when he was 11. One figure reflects the spirit at LAMI: the Smaky 100 core team consisted of less than 10 persons. In contrast, the engineering support for the Apple Lisa consisted of 90 persons.

By then, the time was ripe for the Smaky to leave the walls of EPFL. The Smaky 100 found niches in colleges, offices, libraries. An independent firm, Epsitec, took over manufacturing and distribution. Well, it isn't so independent after all: Epsitec consists mainly of Micro Club members, and Jean-Daniel Nicoud's wife leads it. Rumors persist that many dinners burned while she was busy hacking on her Smaky.

As user numbers increased, so did the software base. About 160 application programs exist today, some of them written by schoolchildren. The programmers divided into two clans, the real-time assembler freaks and the high-level-language application programmers. Smaky supports many languages, like Basic, Pascal, or Lisp; but Modula 2 is clearly the top choice. Students are taught programming in that language and like the well-structured programming style. Connections to Wirth's institute in Zurich also helped.

Is the Smaky a success? In terms of numbers, it isn't. About 1,300 Smaky 100s are in use today (less than the daily production of Apple). This number represents 30 percent of the machines used in college education in the French-speaking part of Switzerland but only 1 percent of the machines installed in the country. With such a small market, the machine costs more than an Atari. But price is not the barrier. A Smaky cannot compete with an IBM PC, which comforts the conservative user's opinions and investments by standards and solid names at the expense of sophistication.

The Smaky is a Tupperware machine: People buy it because somebody reports good experience with it. The users appreciate the possibility of speaking with the programmer, and the programmer welcomes users' comments to improve the product. The users more readily accept imperfections in the software because they know who cares for it.

The Smaky success is mainly a human one: The teamwork in the Micro Club and LAMI created a stock of motivated and experienced programmers—l'ami also means friend. It brought hands-on experience seldom found at universities. Sometimes, after a sleepless night working on the Smaky, Jean-Daniel Nicoud would show up at his lecture quite unprepared and tell the students how to debug boards and which pitfalls to avoid in design. His optional lectures are better attended than many obligatory ones. The message is: You do not need to go to the United States to participate in the adventure of personal computing. This message alludes to those Swiss that became accepted in Switzerland once the Swiss mistook them for Americans—like Gespac or Logitech.

The Smaky 324 (1988): A fully supported, 68020-based personal computer.

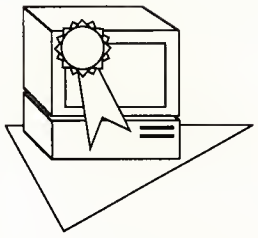
Despite the Cassandra calls, the Smaky story is not finished. The 1988-generation Smaky 324 with its 68020, coprocessors, full-page screen, and 16-Mbyte memory came out at about the same time as the Macintosh II. Software matured and became robust. It is now a fast, multiwindow, multitasking workstation tailored for graphics processing and comparable to the Apollo, but selling at Macintosh II prices. And the schematics for the next machine are already on disk somewhere on a Smaky 324...

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 174 **Medium** 175 **High** 176

Department



New Products

Marlin H. Mickle
University of Pittsburgh

Send announcements of new microcomputer and microprocessor products, and products for review, to Managing Editor, IEEE Micro, *10662 Los Vaqueros Circle, Los Alamitos, CA 90720-2578.*

The latest view of optical disks

Build your own master

PC users can network and store CD-ROM disk data with Discus disks and the Aganet system.

Discus Rewritable optical disks allow MS-DOS and OS/2 users to store, retrieve, modify, or delete data for most applications with storage of 650 Mbytes. Users can also employ the Rewritable Optical Disk Subsystem to create pre-masters in CD-ROM format for proofing purposes. The system also allows users to duplicate master CD-ROMs for cost-effective distribution to small groups of other users.

The Aganet networking system lets Token Ring and Ethernet users simultaneously access a CD-ROM disk without using extensions. **Advanced Graphic Applications; $250 (Rewritable disks); from $4,995 (subsystem); OEM pricing (Aganet).**

Disks **Reader Service Number 12**
System **Reader Service Number 13**
Network **Reader Service Number 14**

Laser disk stores 600 Mbytes

The Laserbank 600 R rewritable optical disk system offers 600 Mbytes of storage to MS-DOS, SCO Xenix, or Novell Netware users. A software interface supports standard file-system manipulation commands. A disk-access time of 95 ms promotes network file serving, medical imaging, CAD, CAE, or CAM.

A related product, the Laserbank 600 CD, includes an MS-DOS interface that emulates standard read-only disks or floppy drives. The CD-ROM optical disk system comes with a host bus adapter for either IBM PC ATs or Microchannel architecture buses. The 350-ms-access system is available in internal half-height or external full-height configurations. **Micro Design International; $6,995 (600 R); from $995 (600 CD).**

600 R **Reader Service Number 10**
600 CD **Reader Service Number 11**

Robotics services "jukebox"

With a capacity for 50 removable, double-sided cartridges, the LF-J5000 storage disk system uses robotics technology to load cartridges into a LF-5010, 5.25-inch optical disk drive. The robotics can also remove a cartridge from the write-once, read-many drive and turn it over for reading purposes. Users can change cartridges through standard computer commands.

A built-in SCSI controller supports MS-DOS, Macintosh, Xenix, and Novell Network environments. The 18.9 × 27.56 × 27.56-inch jukebox offers 47 Gbytes of disk storage and can be mounted on a 19-inch rack. **Panasonic; $40,000 (LF-J5000); from $3,300 (LF-5010, stand-alone).**

LF-J5000
Reader Service Number 15
LF-5010
Reader Service Number 16

Erasable disk stores 652 bytes

A 5.25-inch version of Verbatim's TMO System 35/60's 3.5-inch erasable disk boasts 652 Mbytes of storage and conformity to ANSI and ISO standards. Magneto-optic techniques afford high densities in optically assisted perpendicular magnetic recording.

An optical head writes on the disk by focusing a laser beam on the magnetic film while applying a magnetic field from a bias coil. The absorbed light heats the film under the focused spot, lowering the film's coercivity and enabling the bias field to magnetize that small region of the disk.

The company plans production in the last quarter of 1989. **Verbatim.**

Reader Service Number 17

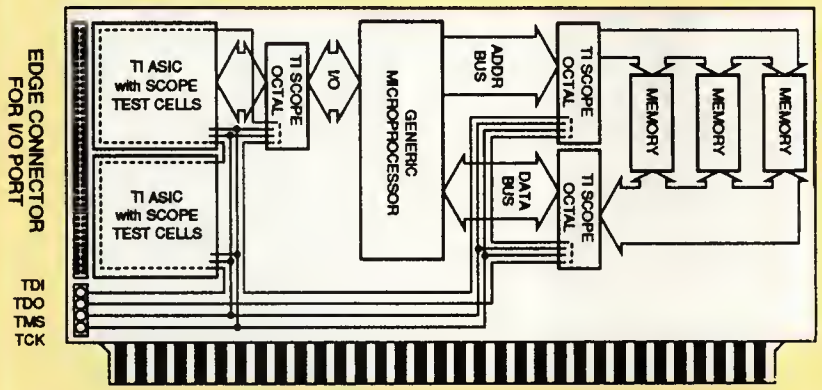
Stand-alone drive stores 900 Mbytes

The WM-S070 disk drive comprises the WM-D070 5.25-inch, write-once optical disk drive, an embedded SCSI controller, and power supply in a 4.4 × 8 × 16-inch enclosure. The unit is plug-compatible with IBM PC AT/XTs, Macintoshes, VAXs, and Sun 3/4 workstations.

A modified-constant-angular-velocity mode provides up to 900 Mbytes of storage and a 2.6 to 5.5 Mbit/s data-transfer rate. Features include support for eight daisy-chained WORM drives and a 10,000 MTBF. Seek times average 90 ms. **Toshiba America; $3,595 (quantity discounts).**

Reader Service Number 18

Cell promotes board-level testability



This JTAG/P1149.1 application shows a multiprocessor-based controller card with I/O control circuitry partitioned onto two ASICs. SCOPE cells added to the ASICs provide boundary-scan capabilities.

The SCOPE supplement to Texas Instrument's TSC500 series of 1-micrometer standard cells contains 14 testability cells. One cell includes a JTAG/IEEE P1149.1 test-access port controller. (JTAG stands for the international Joint Test Action Group.) The other 13 cells in the supplement support input, output, and bidirectional I/O buffers.

SCOPE cells offer ASIC designers board partitioning without adding any devices. Using a series of bit-slice test elements, designers can build structured, hierarchical test systems that use the JTAG/IEEE P1149.1, four-wire, serial architecture.

Designers can add standard test capabilities to systems at a space cost of four pins per device and associated PCB wiring. OEMs can use the cells to increase fault coverage, implement pseudorandom pattern generation, provide nonintrusive system emulation, and develop built-in self-test circuits.

The company states that the SCOPE cell family meets many of the testability requirements of Mil-Std-2165, adding that boundary-scan test methodology detects the electrical problems caused by mechanical stress.

Hardware components, CAD support, computer-aided test tools, and product support are available from the company. **Texas Instruments (free supplement to current TSC500 library).**

Reader Service Number 19

Store a million documents

The Origin database storage system uses artificial intelligence and WORM technology to provide a total capacity of 240 Gbytes, or 140×10^6 document pages. The standard system uses three 2-Gbyte, 12-inch optical disks at a time. An add-on jukebox copes with 20 disks; a Unisys minicomputer interacts with seven jukeboxes. Documents enter through optical character readers, image scanners, keyboards, or transfers from existing databases.

The AI software compiles significant words in a document into a dictionary that acts as an index.

The system can also search for synonyms, antonyms, or related topics. Average search time for all references to a word mentioned 100 times is 0.25 seconds.

The system communicates through an Ethernet network. **Realstream Ltd.**

Reader Service Number 20

No trigger necessary

The Scanplus noncontact image sensor has no moving parts and does not require the operator to pull the trigger. The charge-coupled-device scanner codes UPC bars at a distance of up to 1.00 inch and reads industrial codes from 2.75 inches. Scanplus features on-board decoding and interfacing and connects to OCR and RS-232 ports. A special model connects to the IBM 4683 register. Users can press keys on the terminal while holding the scanner. **Barcode Industries.**

Reader Service Number 21

Go directly to film

The Autobar software program generates bars and alphanumeric characters in one step for typesetting, imaging, and PostScript proofing purposes. Users can incorporate bar codes into page-ready layout with coupon borders, tints, screens, or graphics.

The package contains a code library which allows database-management programs to sort items. Users can view the product before and after it is created.

Autobar allows the creation of both job and format files for typesetting compatibility. Autobar requires an IBM PC XT/AT with 512 Kbytes and DOS 3.0 that is Monochrome, CGA, or EGA compatible. It also comes in a LAN version. **CompWare.**

Reader Service Number 22

Burr-Brown adds a portable terminal

The TM7400 Portable Data Collection Terminal lets users collect data and transmit the resultant file to a host by downloading in batch mode. Application data may be uploaded through the communications port of a host IBM PC or through the auxiliary serial port of a Burr-Brown microterminal connected to the company's integrated architectural network. Users can generate data-collection programs for the 18-ounce terminal on an IBM PC that can either be downloaded into the terminal's RAM or burned into a 32- or 64-Kbyte user EPROM. **Burr-Brown; $1,595 (not including program-development services).**

Reader Service Number 23

Reading, touching, and talking

A better mousetrap?

The Mousetrap touch-screen emulator intercepts a mouse's interrupt calls before they reach the application program and directs them to the touch-screen driver. Users can perform normal mouse functions by touching the screen. Dragging, clicking, pulling down menus, or cutting/pasting an object can be accomplished either by mouse or by hand. Audible signals can also substitute for the clicking of a mouse button.

Mousetrap works with PC Paintbrush, Quickbasic, and Quick C and ships with Accutouch, Duratouch, and Intellitouch screens upon request.

Mousetrap for Windows (also shipped upon request) allows users to resize windows, move objects, and activate commands. It runs transparently after installation into the Microsoft Windows environment. **Elographics.**

Mousetrap
Reader Service Number 24
Window version
Reader Service Number 25

Computer speaks many languages

The speech-activated Voice Computer "learns" to recognize commands and responds with a spoken voice. Applications vary from office and factory automation to telecommunications to language translation.

The computer's base voice-recognition vocabulary comes in banks of 500 words or phrases in six languages—English, Japanese, French, German, Italian, and Spanish. Voice-recognition software understands continuous word or phrase input and interleaved word, connected-speech input. Artificial intelligence recognizes the difference between "2," "to," or "too."

A built-in battery allows 2.5 hours of use; a built-in recharger accomplishes its task in 6 hours. A universal power adapter allows operation from an AC power source. **Advanced Products and Technologies.**

Reader Service Number 27

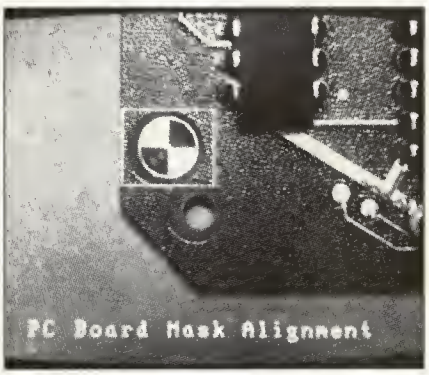
The 3-pound Voice Computer contains a 4-Mbyte memory, three 8- or 16-bit microprocessors, graphics, microphones and speakers, and RS-232 communications at 19,200 bps.

Turn your AT into a machine-vision system

The Itex-align package allows pattern matching, fiducial object registration, alignment, and tracking. The pattern-recognition software works with the PCvisionplus Frame Grabber and the Itex PCplus subroutine library on an IBM PC AT. It utilizes a normalized correlation technique and can automatically "learn" to determine the most discriminant pattern of an object.

Development tools include a menu-driven utility, command-line interpreter, and library of alignment functions. **Imaging Technology; $2,250 (per license).**

Reader Service Number 26



Itex-align software implements PC board fabrication and assembly.

Scanner offers trainability

The Complete Page Scanner allows users to add graphics images to documents and accurately "learns" a new font in several minutes. An interface card and Smartscan software let users crop, erase, rotate, scale, pixel-edit, and convert graphics formats.

When used with the Complete PC's fax family, the 300-dpi scanner provides images directly from PCs to Group III fax machines. Optical character-recognition software—the Complete OCR/Page—reads single-spaced, proportionally spaced, and typeset materials. **The Complete PC; $899 (scanner); $399 (fax); $495 (OCR software).**

Scanner **Reader Service Number 28**
Fax **Reader Service Number 29**
Software **Reader Service Number 30**

Scanning through the windows

The Relisys VM3021 Image Scanner flatbed unit allows line art and halftone modes to be mixed in eight windows on a page. Users can select a maximum resolution of 300 × 600 dpi or choose a 75- to 300-dpi level for either horizontal or vertical dimensions. The unit's 4 × 4 pixels simulate 16 gray scales, which can be stored.

Eight-step brightness and contrast controls clarify scanned line art, halftone photos, text, and mixed graphics. The unit can scan an A4-sized page in user-selectable speeds of 9.9, 16, or 20 seconds. An add-on card provides a bidirectional Centronics host interface.

Basic software functions include scanning parameter setup, or the ability to view images and save them as files. Top Image Editor software for desktop publishing provides image editing, cropping, rotation, expansion, and contraction. **Relisys; $1,495 (VM3021); $195 (TIE software).**

Scanner **Reader Service Number 31**
Software **Reader Service Number 32**

Industrial-strength scanner

The Model RWM1000 I/O scanner has 32 programmable analog inputs, 4 configurable analog outputs, and 32 digital/discrete I/O lines. Complex digital functions include high-resolution frequency and high/low speed-counting input.

Users can select function-and-range software for 32 single-ended or 16 differential input channels. The scanner measures signal sources with either 12- or 14-bit resolution with an update rate of four times per second for all points.

Digital I/O functions are configurable in eight channel groups. Two serial interface ports communicate with a host computer and remote workstation at the same time.

A Timesaver menu-configurable control and acquisition package collects and stores data while it displays in several modes. Menus allow sensor linearization, alarm monitoring, and annunciation. **Industrial Computer Source; from $1,695.**

Reader Service Number 33

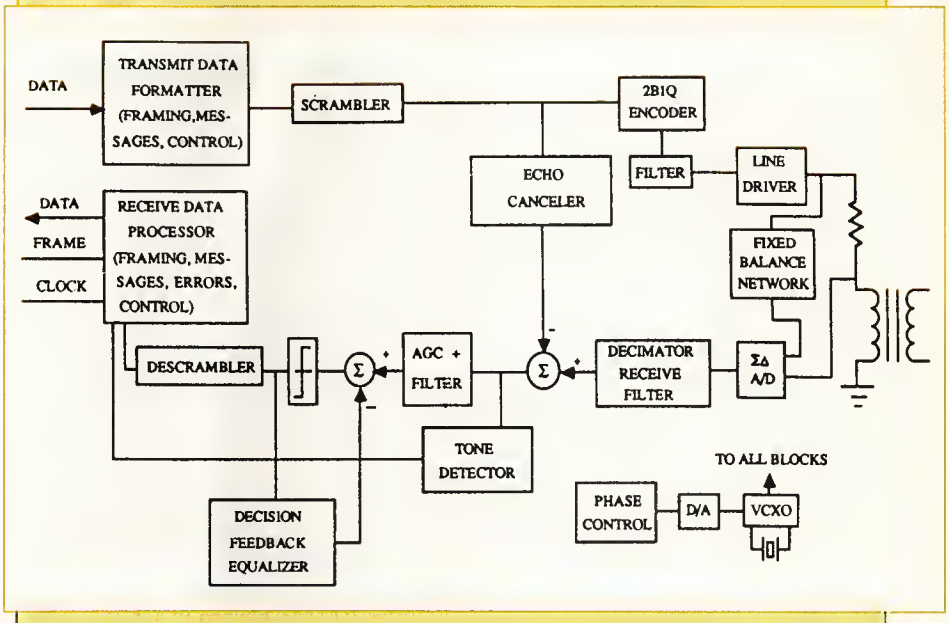
A very touchy screen

According to the company, the Capacitive Touch Screen can handle more than 5 million contacts in any one location because it consists of a glass sheet with a conductive coating bonded to its surface. The system contains one controller and one sensor. Controller options include Serial, IBM PC bus, and Macintosh ADB.

Touch-screen sensors come in 24 sizes for monitors or in flat-panel or custom-sized editions. Snap-on kits for the Macintosh SE and Apple II are externally mounted. Mac 'n Touch screens allow users to open windows, drag icons, and select from menus by touching a screen. Hypercard combinations provide interaction with public-information displays. **MicroTouch; from $330 (standard screen sensors); from $360 (controllers); from $435 (snap-on screens); from $745 (Mac 'n Touch kits).**

Sensors
Reader Service Number 34
Controllers
Reader Service Number 35
Snap-ons
Reader Service Number 36
Kits
Reader Service Number 37

ISDN chip set meets standards



Block diagram of the T7262/T7263 chip set.

The Integrated Services Digital Network chip set was designed to comply with ANSI North American U interface standards. These standards refer to transmission over two copper wires for desktop-terminal and telephone connection to an ISDN switch.

The two-wire, 2B1Q line-code T7262/T7263 chip set provides two 64 Kbit/s bearer channels for voice or data transmission and one 16 Kbit/s data channel for controlling packets and messages. A K2 company interface lets users send and receive voice and data over one analog telephone line without modems. Features include an echo canceller and hybrid for duplex operation, a scrambled data stream, and a nine-symbol, nonscrambled synchronization word-frame heading.

The analog T7262 and the digital T7263 silicon ICs come in 144-pin LCCs for surface-mount packaging with a 5-volt power supply. **AT&T; $95 (sample quantities).**

Reader Service Number 40

Video digitizer adds features

Computereyes Version 3.1 features 640 × 480-point-resolution image capturing and supports 256-color or 64-gray level modes. Additional enhancements include paint-type and desktop-publishing formats and routines to smooth or sharpen images or turn them into halftones.

Captured images can be saved to disk in a number of formats independent of a computer's display capabilities.

The device-driver software allows users to write image-scanning programs into applications for industrial, OEM, and educational markets. **Digital Vision; from $130 ($15 upgrade) (digitizer); $100 (driver).**

Digitizer **Reader Service Number 38**
Driver **Reader Service Number 39**

Video and computer graphics merge

The Spectrum NTSC provides live pictures with VGA compatibility and can overlay a series of graphic formats. It also operates as a frame grabber that can digitize both video files and frames in real time in 16.8 million colors.

The system simultaneously displays both graphics and video on one screen. It accepts composite video from any National Television System Committee (NTSC)-compatible source and outputs the digitized image to a VCR or monitor. Users can program the picture location on the screen and size pictures by combining zooming factors and screen sizes. **Redlake.**

Reader Service Number 41

Memory developments

Programmed in small lots

Because these plastic DIP devices are custom-programmed as part of each wafer's normal test process, the company states they are cost effective in lots as small as 5,000.

The EspressROM family offers densities from 64 Kbytes to 1 Mbit and speeds from 100 to 250 ns. The company plans plastic LCC packaging in the third quarter. **Advanced Micro Devices; $4.25 (250-ns 27X512) (10,000s).**

Reader Service Number 42

It's all in the packaging

A windowed, ceramic leaded chip carrier package for 1-Mbit CMOS EPROMs features low susceptibility to thermal mismatches as well as erasability and reprogrammability in the early stages of surface-mount designs. The company claims pinout and footprint compatibility with plastic LCCs eliminates circuit board redesign in later stages.

The EPROM comes in 120-, 150-, 200-, and 250-ns versions with ROM-compatible and JEDEC-standard pinouts with 32 or 44 pins. Company claims a 15-percent cost savings over LCCs. **Mitsubishi; from $36.50 (100s).**

Reader Service Number 43

Sequential-access EPROM

The 256-bit MB8541 CMOS EPROM provides an on-chip address counter that is automatically incremented by clock input. Power supply can vary from 3 to 8 volts while operating temperatures range from –40° to +85°C. The three-state input device also includes 64-bit cells to service incoming test data and store identification code.

In addition to conventional EPROM programming, users can electrically program the MB8541 with one 9-ms pulse. Packaging options include 8-pin plastic DIPs or plastic flat packs. **Fujitsu Microelectronics; $1.95 (10,000s).**

Reader Service Number 44

EPROMs store 4 million bits

A trio of high-density memory devices includes the 256-Kbyte × 16-bit 27C240, the 128-Kbyte × 16-bit 27C220, and the 256-Kbyte × 8-bit 27C020.

The nonvolatile 27C240 stores 4 Mbits and comes in a JEDEC-standard, 40-pin CerDIP in 150- and 200-ns access versions. It is a pin-compatible upgrade of the company's 40-pin, 1-Mbit 27C210.

The second two EPROMs feature 2-Mbit densities. The 27C220, packaged in a 40-pin CerDIP, provides compact board design for PCs and is also pin-compatible with 27C210.

The 27C020, housed in a 32-pin DIP, supports embedded systems that use multiple EPROMs for mass storage. This device is a direct socket replacement for the 32-pin, 1-Mbit 27C210. **Intel; $100 (27C240); $39 (27C220); $35 (27C020); (all in 10,000s).**

27C240 **Reader Service Number 45**
27C220 **Reader Service Number 46**
27C020 **Reader Service Number 47**

EPROMs gain density

The NMC27C512A 1.5-micrometer, 512-Kbit, erasable memory device with a 64 Kbyte × 8-bit configuration offers access times that range from 150 to 250 ns. It comes in a 28-pin DIP with a transparent lid so that ultraviolet light can erase the bit pattern.

Also available is the 1-Mbit, 128-Kbyte × 8-bit NMC27C010 with times that range from 200 to 250 ns. Clocked sense amplifiers boost access times. The CMOS EPROMs include extended and military versions and combine a 110-milliwatt power consumption with 5-volt operation. **National Semiconductor; $7.15 (100s) (250-ns NMC27C512A).**

Reader Service Number 48

Device peps up access times

The V63C64 static RAM features 25-ns access and 9-ns output enable times. The 8 Kbyte × 8-bit SRAM for cache-memory applications in 80386-based PCs and workstations interfaces with Austek's A38125/A28285 cache controllers, the Intel 82385, and the M68000 family. Two chip-enable inputs support memory expansion.

The CMOS device can also serve as a conventional SRAM for printers, modems, and graphics. Parts come in 300-mil, plastic DIPs. **Vitelic; $17 (volume quantities).**

Reader Service Number 56

Low-power, high-speed memories

Designed for cache memory, writable control store, and data-buffer applications, the VT62832 SRAM features 300 mW active, 100 µW standby, and 15 µW CMOS-standby power use. The VT62832L offers unspecified lower rates.

The 256-Kbyte VT62832 features access and cycle times of 35 ns. Organized as 32,768 words × 8 bits, the SRAM comes in 300-mil plastic DIPs and SOJ packaging. **VLSI Technology; $75 (PDIP VT62832); $86.25 (SOJ VT62832); $82.50 (PDIP VT62832L); $93.75 (SOJ VT62832L); (all in 100s).**

PDIP VT62832
Reader Service Number 49
SOJ VT62832
Reader Service Number 50
PDIP VT62832L
Reader Service Number 51
SOJ VT62832L
Reader Service Number 52

Flexible SRAMs access data in 30 ns

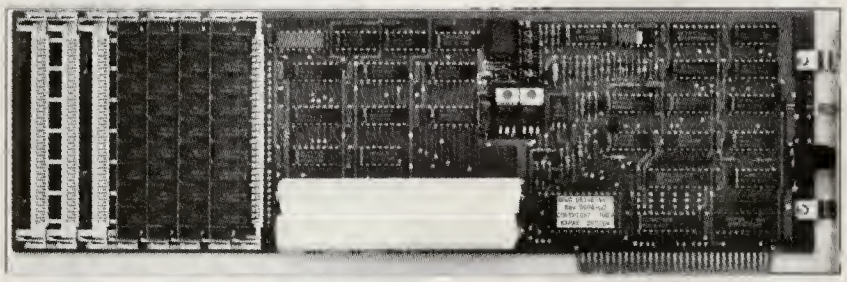
The MCM6264 fast static RAMs offer 30- to 70-ns access times in an array of 300- and 600-mil plastic DIPs and 400-mil, small-outline, J-lead packages.

The company also offers the MCM6206 fast static RAM with 32-Kbyte × 8-bit memory for byte-wide organization. The company claims no-wait-state performance for most microprocessors. Sample quantities are now available. **Motorola; $17.75 (MCM6264, plastic DIPs) (100s); $20.89 (MCM6264, SOJs) (100s); $60 (MCM6206) (production quantities in third quarter of 1989).**

6264 PDIPs
Reader Service Number 53
6264 SOJs
Reader Service Number 54
6206
Reader Service Number 55

Advances in disk emulation

Emulation speeds access



The Novo Drive 8000 features plug-in SIMM memory modules that supply up to 8 Mbytes of storage.

The Novo Drive 8000 disk emulator for IBM PC/AT/XTs and PS 2/30s is implemented with semiconductor memory on a printed circuit card with solid-state design. RAM-inherent data access reaches speeds the company claims are 8,000 times faster than those for mechanical drives. System interface logic with DMA moves data each memory cycle.

A separate AC power adapter maintains data when the computer is off. The drive can be physically transferred to another system without data loss. The drive is suited to such operations as frequent disk transfers and programs that use overlays or templates. **Kapak Design; $375 (without memory modules).**

Reader Service Number 57

Thincard family grows

Two solid-state disk drives that can fit into the same-size, sealed, gasketed housing as the original IBM PC Thincard system have been introduced. The first is an exact plug-in replacement for the Dallas Semiconductor DS1217M nonvolatile RAM cartridge and DS9020 Cartridge Clip. The second is an OEM drive unit.

The TCD DAL/1 Dallas cartridge unit plugs into a 28-pin JEDEC SRAM socket. It operates by switching 32-Kbyte banks of RAM through decoding sequences on the address bus. User-configurable switches and headers allow operation in single-cartridge mode or recognition of a selected cartridge clip position. One cable accommodates up to 16 drives.

The TCD/2 OEM drive contains buffering and switching circuitry specified by card manufacturers and address latches for high-order address bits. The directly addressable 256-byte card supports self testing. Coordination of signals with pins on the interface connector facilitate EPROM card programming.

The solid-state drive units support Epson and Mitsubishi IC memory cards (users should specify). **Databook; $195 (TCD DAL/1, 1,000s); $75 (TCD/2, 1,000s).**

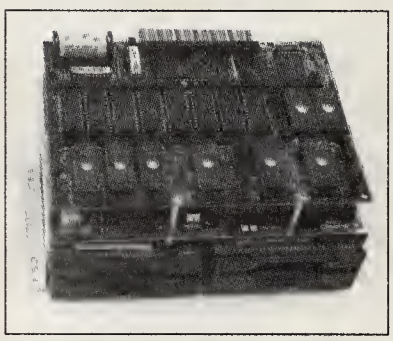
Cartridge **Reader Service Number 58**
Drive **Reader Service Number 59**

Little boards expand memory

The Little Board one-board systems have gained a memory expansion board that adds up to 8 Mbytes of EPROM for solid-state disks (SSDs). (When Little Board/PCs and /286s operate as embedded controllers in small spaces at extreme temperatures with high vibrations, floppy or hard-disk drives do not suffice.)

SSD Expansion Boards conform to the small Little Board form factor and provide sixteen 32-pin, byte-wide memory sockets that can hold a variety of 8- to 256-Kbyte EPROMs and 8- to 128-Kbyte SRAMs.

A number of possible memory configurations result from organization in three groups of four, four, and eight sockets. An optional on-board lithium battery backs up static RAMs and creates a nonvolatile RAM SSD. **Ampro Computers; $126 (OEM, 100s).**



The SSD Expansion Board, shown straddling a Little Board/PC and two 3.5-inch disk drives, can implement a 2-Mbyte solid-state disk in a space smaller than that occupied by a half-height, 5.25-inch disk drive.

Reader Service Number 60

Microcomputer scans in most environments

The Laser-Wand noncontact scanning system provides a field width of 11 inches, a working distance of 20 inches, and a speed of 36 scans per second.

The self-contained, 21-ounce Laser-Wand allows one-handed bar code scanning. Scanning does not interfere with the LCD display or keyboard functions. The system includes an 8-bit microcontroller, CMOS RAM expandable to 1 Mbyte, a 33-key alphanumeric keyboard, and a 32-character backlit display.

The 64-Kbyte EPROM scanner, programmable in Universal Data Language, discriminates between bar code symbols automatically and interfaces to RS-232 devices. **Hand Held Products.**

Reader Service Number 61

Faster and more RISCy processors

ECL boosts processor speed

According to the company, emitter-coupled-logic implementation produces the 10486's 30 native MIPS with a 90-MHz clock. The 32-bit RISC for high-resolution graphics, embedded controllers, robotics, and optical/voice recognition includes an on-chip MMU and 33-ns memory cycles.

Development systems and software tools are available for the system, which supports Basic, C, Cobol, Forth, Fortran, and Pascal programming languages.

The 10486 directly drives I Mbyte of high-speed static or dynamic RAM without support chips and provides memory bandwidths of up to 60 Mbytes/s.

The processor comes in a I49 PGA package and is also available in a military version. **Integrated Digital Products; $895 (100s).**

Reader Service Number 62

Just another one-million-transistor machine!

The 32-bit i486 microprocessor provides 20 VAX-equivalent MIPs at clock speeds up to 33 MHz. Features include the instruction sets of the 386DX microprocessor and the 387DX math coprocessor. The one-million-transistor machine (a state it shares with the company's i860 CPU) also integrates a paging and memory-management unit and an 8-Kbyte data and instruction cache. Pipelining and RISC design techniques execute frequently used instructions in one clock cycle. A burst data-transfer mechanism allows four 32-bit words to be read sequentially from memory.

The company states that at 25 MHz the processor executes 37,000 Dhrystones/s and 6.1 double-precision million Whetstones/s. At 33 MHz, Dhrystones/s performance reaches 49,000 and the million Whetstones/s increase to 8.2.

Multiprocessor instructions and hardware cache-consistency protocols aid the implementation of multiprocessor systems. Current packaging is in I68-pin PGAs. **Intel; $950 (1,000s).**

Reader Service Number 63

Counting on the Abacus

The Abacus 4167 floating-point coprocessor plugs into a socket on 80486-based system boards to run computationally intensive applications. Existing applications that support the Abacus 3I67 for 386-based computers also support the Abacus 4167.

A I42-pin PGA socket designed onto the system board provides signals necessary to interface the 4I67 to the 486. Memory-mapping allows most interface signals to connect to the 486's data and address buses.

The 4I67 is upwardly software-compatible with the 3I67 and features sixteen 64-bit registers for performing calculations without transfering data between the floating-point unit and memory. A 64-bit floating-point data path includes an ALU, a multiplier, and a divide/square root unit.

The company states that—all things considered—the Abacus 4I67 offers RISC-level performance to CISC users without the need to develop application accelerator boards. Sample quantities will be available in September, production quantities in December I989. **Weitek; $565 (1,000s).**

Reader Service Number 64

Transputer adds speed and functions

The IMS T425 32-bit transputer integrates a I2.5-MIPS microprocessor, four serial links running at 2.4 Mbytes/s, 4 Kbytes of SRAM, and a 32-bit memory chip. It also offers an additional set of instructions and new pin functions. A refresh-pending pin holds DMA requests, and an event-waiting pin allows users to control external logic.

The transputer can function as a conventional processor, form multiprocessing networks/arrays, and spark embedded-control applications. It is available in 84-pin PGA packaging in I7-, 20-, or 25-MHz versions and is pin-compatible with the T414 32-bit transputer. **Inmos; $269 (100s).**

Reader Service Number 65

Virus protection for the Mac

The Symantec Antivirus for Macintosh (SAM) program consists of an Intercept initialization component and the Virus Clinic application.

When Intercept identifies illegal actions, the user can let them occur, stop them, or tell SAM to learn the activity for future repetition.

The Virus Clinic detects known viruses within the Mac and lets the user delete the file or helps to make repairs.

The company claims that SAM can stop Scores, Nvir, Hpat, Init 29, and Anti viruses. Macintosh System Version 4.2 and Finder Version 6.0 are required. **Symantec; $99.95.**

Reader Service Number 66

Scanner fits in your pocket

The 6-ounce Timewand II bar code scanner for inventory control, asset tracking, and gathering remote-site data features a 32-character display and a 19-button keypad. Memory size distinguishes the 32-, 64-, or I28-Kbyte versions.

The metal-encased reader contains an intelligent battery recharger and provides both scan sequencing and cross-referencing for data organization. The $4.1 \times 2.6 \times 0.6$-inch Timewand II contains an RS-232 serial port. **Videx, Inc.; from $698 (Timewand II); $18 (recharger kits); $29 (cable for IBM or Macintosh); $380 (software for IBM or Macintosh). (Volume discounts.)**

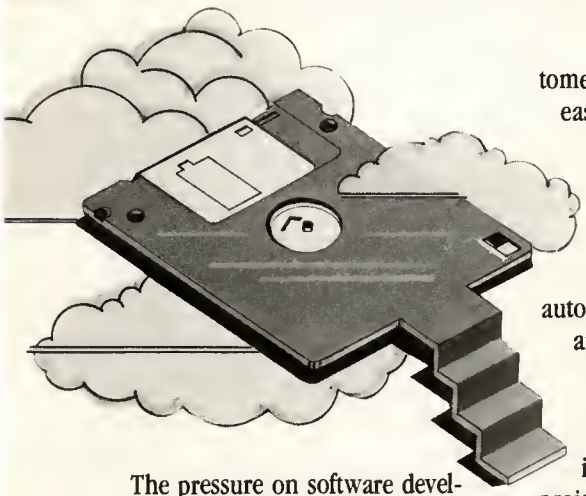
Timewand **Reader Service Number 67**
Kits **Reader Service Number 68**
Cable **Reader Service Number 69**
Software **Reader Service Number 70**

Reader Interest Survey

Indicate your interest in this department by circling the approriate number on the Reader Interest Card.

Low I80 **Medium** 181 **High** 182

Your Stairway To Software Heaven.



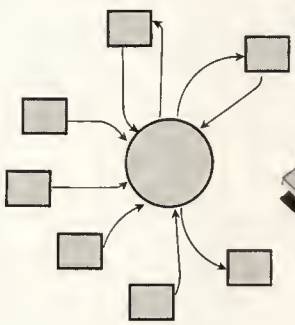
The pressure on software developers to *produce* has never been greater. Yet there they sit, often reinventing the wheel, with more computational power than ever at their fingertips and the clock ticking away. Product delivery deadlines? So much pie in the sky.

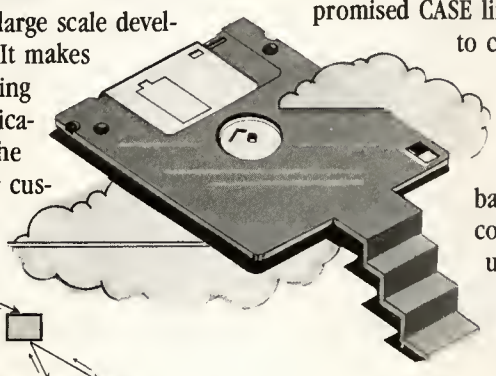
The problem? How *best* to put all that PC CPU capacity to good use.

The solution: the **Visible Analyst Workbench.**

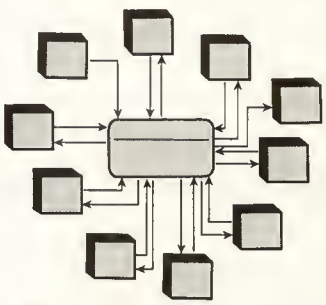
The Visible Workbench makes the full power of CASE accessible to *everyone*. Running as a multi-user tool on Novell LANs or on individual PC workstations, the Visible Analyst Workbench lets teams of software engineers work together — and *simultaneously* — on large scale development projects. It makes after-the-fact piecing together of specifications a thing of the past. And, as our customers delight in telling us, it's so easy to learn and use that people begin working more productively on "day one."

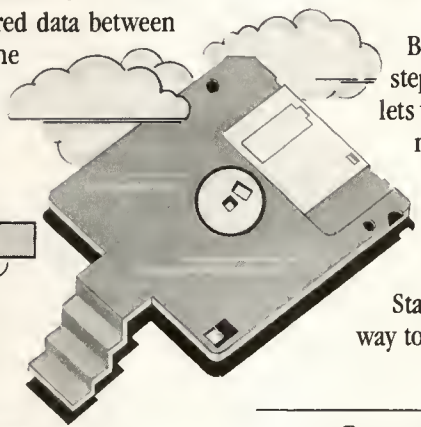
The Visible Analyst Workbench delivers *real* development power. The power of automated, linked structured analysis and design. The power of an automated data repository. The power of prototyping. The power of instantaneous communication and shared data between project members. The power of accurate, validated high level specifications with full documentation. And, soon, bridges to code generation, completing the promised CASE link "from pictures to code".





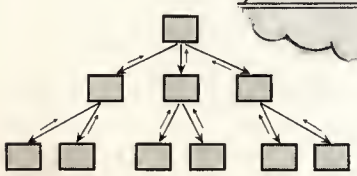
In fact, the Visible Analyst Workbench is the *only* PC-based CASE tool that combines ease of use, self-implementation, cost effectiveness and *true* multi-user capabilities. And, because it *is* a CASE tool, its usefulness will seem everlasting.





Best of all, our step-by-step product growth path lets you begin building internal CASE resources at down-to-earth prices starting under $300.

The Visible Analyst Workbench. Start building *your* stairway to software heaven today.



Down to earth prices

Professional Series — For large, multi-project systems development:

3-Node LAN Pak	$ 3,500
per additional node ..	700
Stand-alone version ..	1,785
with Prototyper	2,380

Personal and Educational Series — For small project development and educational needs.

Personal Edition	695
Educational and Training Version	295

Visible Systems CORPORATION

The Bay Colony Corporate Center • 950 Winter St. • Waltham, MA 02154 USA
(617) 890-CASE FAX (617) 890-8909 Telex 261102 VSCUR

Reader Service Number 1

Product Summary

Marlin H. Mickle
University of Pittsburgh

For more information, circle the appropriate Reader Service Number on the Reader Service Card.

MANUFACTURER	MODEL	COMMENTS	R.S. NO.
CASE products			
Multiprocessor Toolsmiths	Caseworks/ RT tool set	Development environment for multiple processor real-time and/or embedded systems contains tools for first analysis, design, coding, integration/testing, and maintenance. Features upgradable components and modular expansion from MS-DOS, Unix, and VAX/VMS systems.	80
Oracle Corporation	CASE*generator tool	Development software automatically generates portable applications from design specifications. Translates database-table and program-module definitions by using the company's SQL Forms and CASE*dictionary. Applications contain support for lists of valid values, help and hint text, and automatically synchronized data from multiple database tables.	81
Integrated Systems	Autocode/ matrix Version 2	Real-time-control development software integrates CAE and CASE of design, analysis, simulation, and code-generation functions into one workstation environment. Automatically generates military-specification 2167A documentation. Features include an object-oriented database and asynchronous triggered systems capability. Available on VAX, Sun, and Apollo workstations. From $25,000.	82
Interactive Development Environments	Software through Pictures tools	Family of development products features an open architecture that lets multiusers extend and customize their environments. Aids in the analysis, design, and documentation stages of software development and runs on Sun 3/80 and 3/400, Sparcstation 1 and 330, and Sparcserver 330 workstations.	83
Quicktek Corporation	Schooner environment	Data-driven, object-oriented engine allows users to design, document, and run an application without generating code. Users can link the company's Clipper, C-language, and ASM routines. Built-in design- and run-mode languages reconfigure Schooner itself and let users change the behavior of objects. Third-party vendors can create graphics, communications, and real-time interfaces. $695 (plus shipping and handling).	84
Texas Instruments	Information Engineering Facility tool set	Version 4.0 of CASE environment features version control, workstation prototyping, and a batch-target approach. Enhanced data modeling includes support for subject areas, entity types and subtypes, relationship aggregations, relationships, and partitioning—in one diagram. Users can employ the matrix processor to define object types and matrices.	85

MANUFACTURER	MODEL	COMMENTS	R.S. NO.
Cadre Technologies	Teamwork tool set	Tool kit includes the Architecture Design and Assessment System simulation tool set for co-designing software and hardware. Teamwork also offers the Ada Source Builder code-generation tool. Programs run on Sun-3 and -4 series, Sparcstation 1 and 330, and Sparcserver 330 workstations.	86
Networks			
Datatech	Telan LAN	Network offers file transfers, electronic mail services, and shared printing and disk space to IBM PC AT/XTs. The NetBIOS-compatible system allows network servers to double as workstations. Half-size PC-slot cards combine with memory-resident utility programs to form the network. A password system protects file and resource usage. £390 (two cards and software).	87
Network and Communication Technology	Multiserver	LAN contains up to four 20- or 25-MHz, zero-wait-state, 386 servers; control panel; keyboard; standby power supply; and two dual monitors in one cabinet. Each server includes 11 disk drives and a 275W switching power supply. Users can rack-mount access units, modems, and patch panels. Company also offers LAN CAD network design and resource-management software that runs on MS-DOS 3.0 and requires 2 Mbytes of RAM.	88
FTP Software	PC/TCP software	PC software that implements Sun Microsystem's Network File System protocol supports Ethernet, Star LAN, and Token Ring networks. PC/TCP, an MS-DOS version of Transmission Control Protocol/Internet Protocol (TCP/IP), allows users to transfer files, transmit electronic mail, and access minicomputers and mainframes. It also performs remote tasks on multivendor computer systems. Includes emulators for VT100, VT220, and IBM 3270 terminals; Berkeley sockets; remote backup; domain name resolution; and a NetBIOS option. $490 (with applications).	89
3Com Corporation	Netbuilder IB/2000 and IB/2001 routing bridges	The Netbuilder family of hardware platforms uses an MC68020 CPU to forward data at 10,000 packets/s. Internetwork bridge IB/2000 supports two-way connection to thick Ethernet cabling, while the IB/2001 includes thick and thin cabling. Both bridges feature system-level network management, custom filters, and source-explicit forwarding for security. A Spanning Tree Algorithm intelligently selects paths and improves network operations. $5,250 (IB/2000); $5,650 (IB/2001).	90
Banyan Systems	Vines Applications Toolkit	Tool kit develops Banyon Vines Version 3.10 network integrated applications. Features include a Unix environment, the Streettalk naming system, and the Mail Gateway application programming interface. A serial-line interface and network-compiler utilities are included. The environment also contains Microsoft Version 5.1-compatible libraries for small, medium, and large memory models and utilities that convert text-file formats to and from Unix and DOS. $1,995.	91
Server Technology	Easy Print printer-control network	Version 2.0 allows users of IBM PC XT/ATs or PS/2s to share any configuration of 42 laser, dot-matrix, and letter-quality printers and plotters. Users select printers—including dedicated printers attached to one PC—from pop-up menus. The software requires 44 Kbytes of memory, an intelligent multiline network-access unit, and modular six-wire telephone cabling. Optional Postscript emulation lets users intermix Laserjet and Postscript print jobs. $399.95 (software for four PCs, network control unit, and four 30-foot cables).	92

Product Summary

MANUFACTURER	MODEL	COMMENTS	R.S. NO.
Network Software	Adapt SNA 802.2 series software	Connectivity software lets PCs communicate directly with IBM mainframes via multiple Systems Network Architecture protocols. An IBM token-ring interface card directly connects a 3174 controller to the token ring LAN and decreases PC-to-host transfer times. No gateway PC is necessary. Users currently using Adapt SNA packages for synchronous data-link control, coaxial, or asynchronous applications can upgrade to the 802.2 LAN driver without modifying applications. From $245 (five packages).	93
Racore Computer	SL-80 and SLE-80 LAN stations	Company Norton-utility benchmarks indicate the workstations perform 23 times faster than IBM PC XTs. Page-interleaved RAM, a 20-MHz 80286 chip, and system/video BIOS execution in RAM rather than ROM contribute to this speed. Both IBM-compatible stations are optimized for Novell Netware and support the 80287 math coprocessor. The diskless SL-80 provides two full-length, 8- to 16-bit expansion slots; the single-disk SLE-80 offers four. From $1,779 (SL-80); from $1,979 (SLE-80).	94
Advanced Micro Devices	Ethneval 5 Ethernet adapter card	According to the company, the IBM PC AT adapter card has achieved a Novell data-throughput rating of 271 Kbytes/s (single station) and 956 Kbytes/s (maximum network). The half-card chip set consists of the Am7990 Lance, the Am7992B SIA, and the Am7996 transceiver. The company offers manufacturing rights. $495 (evaluation kit).	95
Standard Microsystems	COM90C65 LAN controller	Third-generation device integrates an ARCnet controller and transceiver with circuitry to interface a PC bus. IBM PC AT add-on boards require only four ICs rather than 25 MOS/LSI devices. $17.50 (5,000s).	96
Thomas-Conrad Corp.	Thomas-Conrad Networking System	The TCNS provides a 100 Mbit/s data rate, 10,000-foot workstation separation, 20,000-foot network span, and support for 255 stations. The hardware-implemented system is compatible with Novell's Netware, Banyan's Vines, and Western Digital's Vianet with ARCnet drivers. Conversion of existing ARCnet LANs involves hardware replacement but no software modifications. (Late-1989 availability).	97
Chips			
Intel Corp.	85C508 address decoder	Programmable CMOS, 7.5-ns logic device speeds the microprocessor-to-memory interface in 386TM and 386SXTM microprocessors. A proprietary architecture integrates decoding and latching on one chip. The 28-pin, plastic DIP PLD contains 16 direct inputs, a simplified NAND product-term array, and eight transparent output latches driven by a global latch enabler. $12.50 (10,000s).	98
Advanced Micro Devices	Am29C660D EDC	According to the company, this 32-bit IC detects 1-bit errors in DRAMs in 12 ns and corrects them in 18 ns. Adaptable to 64-bit memories, the 68-pin PLCC, PGA, and QFP chip consumes 0.28W of power. $92 (1,000s).	99
VLSI Technology	VGT200M ASIC	The 1.5-micrometer military ASIC library for Mentor and Daisy workstations supports up to 54,000 usable gate designs, 60-MHz clock speeds, and 2- to 16-mA programmable output drives. The library is compatible with the company's logic synthesis and compiler workstation Interface software and includes 280 small- and medium-scale integration functions.	100
Intel Corp.	5AC324 logic device	Erasable PLD contains 24 macrocells and provides 50-MHz, pipelined performance for microprocessors and microcontrollers. The CHMOS device at twice the density of the 5AC312 maintains a 30-ns propagation delay and contains 10 programmable inputs and 24 configurable I/Os. Users can define macrocells allocated with zero to 16 product terms. From $24 (1,000s, US only).	101

Advertiser/Product Index

FOR DISPLAY ADVERTISING INFORMATION, CONTACT:

continued from p. 10

tionable even though the technique is not essential and was not dictated by human factors analysis at the time it was created.

In addition, intellectual property protection of screens can, at least in theory, impede efforts to bring about standardization. Standards may be regarded as formally, officially, or institutionally agreed-upon conventions. They are distinguished from those conventions that evolved by informal, unofficial means (de facto industry standards or mere unofficial conventions).

In general, standardization benefits individual users because it allows them to transfer their knowledge of how to use one system to another system. Standardization also benefits corporate users, because it lessens training time and expense, increases worker productivity, and decreases expenses caused by mistakes. Finally, standardization benefits screen designers because it resolves certain design issues and, in effect, thereby interdicts reinvention of the wheel.[9]

Inhibiting standardization efforts could increase consumer confusion over how to use computer programs, and thus have a negative effect on software progress. An official of Microsoft Corp., a leading US software publisher, has asserted that the user interface is ". . . the wrong place to differentiate your product. We in the computer business benefit from the more users who have ready access to the technology. The more confusing we make it for users, the slower the market is going to grow."[10]

The moving target problem

The problem raised by the prospect of protecting utilitarian aspects of screen designs is made more complex, unfortunately, by another fact. What is good design practice, convention, or de facto industry standard is not static. Good design practice is a moving target. Hence, today's individual "expression" in screen design may be tomorrow's routine or standard industry practice, which is to say unprotected "idea" rather than protected expression for copyright law purposes. There is always a first time for anything.

Quit and Save were mentioned earlier as examples of conventional terms for the commands usually so designated in computer programs and their menus. Thus, they are typical public domain features in a menu or similar display.

The moving target problem: Today's "invention" is tomorrow's convention or standard.

Although the fact is probably lost in the mist by now, presumably somebody was once the first to use those terms. By the same token, somebody must have been first to use the highlighting technique discussed above, and every other screen design convention.

The validity of this proposition is not difficult to establish. Consider two command terms involved in the recent Softklone litigation over the main menu screen displays of the Crosstalk and Mirror programs. The menu of the Crosstalk program used **XM**it as the Transmit command and used **RQ**est as the Request command. Here the capitalized and highlighted first two letters of XMit and RQest appearing on the screen display indicated the keystrokes (<XM> and <RQ>, respectively) for invoking those commands. The designers of Crosstalk could not have used T**R**ansmit and **RE**quest as terms in the screen display, because other necessary commands or parameters also began with the same two letters (for example, **RE**ply). Some substitute had to be devised. Counsel for Crosstalk's owners (referring to **RE**ply, **RE**quest, and **RQ**est) contended that the use of the same such commands in Mirror was copyright infringement. "It was precisely this type of idiosyncratic design which represented ... 'extensive original human authorship' and the basis for copyright protection."[2]

Apparently, the plaintiff in the Softklone case was the first user of RQest as an abbreviation for Request. XMit, however, is different. I recall seeing the use of and using the letter *X* as the abbreviation for *trans*, in terms such as Xfrmr, Xfer, and Xistor during the 1950s. But Xmit tests the principle, for surely somebody must have been the first to use it. Certainly, the plaintiff was not the first to use Xmit to mean Transmit. Or if by some accident the plaintiff was first, the usage was an obvious extrapolation from all of the other "trans ... " words in which *X* was long used to abbreviate *trans*.

One response to this commentary might be to conclude that RQest is protectable and XMit is not. The court in the Softklone case seems to have agreed with that approach.[6] The Softklone court found that some commands were represented with conventional abbreviations, but "other choices of symbols are clearly original, e.g., 'RQ' for the 'request' command . . . ." The court considered this fact as supporting copyrightability.

But that approach is unsound and superficial. R* may become the de facto standard abbreviation (and, by the same token, <R*> the standard keystrokes) for command words of the form Re* . . . . (The asterisk is used here as a variable to symbolize the third letter of the word.) This abbreviation could become standard in the next few years, just as *X* has become a standard abbreviation for the prefix *trans*. If R* and <R*> did become that kind of standard, should anyone who used that form of abbreviation on a screen for ReDial, ReFormat, ReLay, ReMote, RePeat, ReXmit, and so on, be considered to infringe on the rights of the first user? One must accept the fact that somebody really was the first to use Quit and Save as commands, and thus as aspects of a user interface. One must also accept as fact that similar important such "firsts" will occur again. The problem is not unique to RQest. It is not ephemeral. It is recurrent.

The moving target problem, therefore, is that today's "invention" is tomorrow's convention or standard. According intellectual property protection to the first user of a useful aspect of screen design may soon afterwards prevent others in the field from employing a useful convention.

The speed of change in the software field relative to the more conventional subject matter of intellectual property law heightens the problem. The average life of a copyright is 75 years, and the life of a patent is 17 years. In the case of books and machines, those periods of protection do not unduly interfere with the creativity of others, or at least there is little protest.

Contrast those time spans with those experienced in the computer software industry. The technique of highlighting on the screen the part of a command or parameter that contains the keystrokes to be used in entering the command or

parameter (and probably every other important screen design technique) had to have been originated by somebody within the last 75 years. (This is the life of a copyright.) There were no screen displays before that time. The technique may well have originated within the last 17 years (the life of a patent), since microcomputers have been in common use for less than that length of time—only since the late 1970s.

The "invention today, convention tomorrow" principle means that according traditional intellectual property protection to particular new, useful techniques in screen design could prove to be imprudent and destructive of competition. It could have a negative effect not seen in other useful arts. Product life and the pace of innovation in software make the duration of traditional intellectual property protection relatively greater in the software field than in other arts. Thus, a type or amount of protection that does not retard technological progress in other arts might retard technological progress in software. This is true at least if that protection is applied to individual components of screen design, such as particular techniques, rather than only to large, entire software products.

The examples of highlighting and RQest, in the Softklone litigation, illustrate the risk. They indicate that some courts may improvidently find copyright infringement where a defendant has taken individual screen design components from the work of a firstcomer in a field. The result could be to impoverish the repertory of screen display techniques needed by workers in the field.

Two additional factors complicate the analysis of emerging conventions for user interfaces. First, sharp boundary lines don't exist along the functionality spectrum as one passes from category to category. These categories range from a) necessary techniques of screen design that are objectively dictated by human physiology or hardware capabilities; to b) IEEE, ANSI, or similar standards; to c) accepted conventions; to d) emerging conventions; to e) mere idiosyncrasies of individual designers without functional or other public significance. Some of these categories, other than category a), will include screen display expedients that are not objectively dictated at the time that they are created. Over time, however, design features in categories b) through d) may become as functional as if they were in category a). That would occur because of user habituation to widely accepted user interfaces.

At any particular time, it may be difficult to classify a particular design technique. For example, it is fair to say that most electrical engineers will recognize *X* on a menu as a prefix meaning *trans.* Most users will recognize a high-intensity, capital first letter of a word on a menu as indicating that the letter is the keystroke for the function, parameter, or command represented by the word. These have emerged as conventions. Use of </> (slash) to invoke a command mode is probably an emerged or emerging convention. It is unclear whether a feature such as <R*> as the keystrokes for a command word of the form "re* . . ." is even an emerging convention.

We must strike a balance between encouraging software innovation and fettering potential innovators.

Uncertainty about where the moving target of a possibly emerging convention is located creates problems. One problem lies in deciding whether legally protecting a technique will preempt a functional, utilitarian screen design expedient. Such uncertainty also raises another question. Should the law start out protecting the feature and then withdraw protection if and when the feature becomes functional because of enough user habituation to it to make it an emerging convention? Or, would it be better to deny protection until it is clear that the feature was not going to become a convention? Each approach has defects.

A second problem concerns whether it is sound policy to have the law create a legally protected interest for screen display and software proprietors in their customers' investment of time and effort in learning to use an interface. We all have seen advertisements of the form, "If you know how to use Program A, you already know how to use Program B. They work alike." Is it better that whatever interest there is in user self-education about, and consequent habituation to, a user interface be considered part of the public domain? Or is it better policy to concede that interest to creators of user interfaces, in order to promote such creativity? Where along the spectrum from a) to e), as described earlier, do we want to put the boundary line of legal protection? This is not a zero-sum game between firstcomer marketers and clonemakers. Users have interests, also—in encouraging innovation and progress, in fostering competition, in minimizing their learning costs, in allocating their personal memory resources efficiently, and in not being exasperated.

Keystrokes and interfaces

These concerns and the same principles extend beyond screen design techniques, as such. Use of particular keystrokes for certain purposes is also conventional in user interfaces and menus embodying them, although not essential. (Such purposes might include invoking a command set, pulling down a help menu, changing to another mode, escaping to DOS, or changing foreground programs.) Copyright protection of such keystrokes could have a negative effect on software progress.

To be sure, the choice of keystrokes may properly be considered to be part of the user interface and not part of the screen display, and therefore not protected by copyright. But typically, somewhere associated with the program is a menu or chart that shows the keystrokes. Thus, the keystroke aspect of the user interface will be embodied in a screen display that is potentially protectable under the copyright laws. The keystrokes may receive copyright protection via protection of the menu, so that the copyright laws, in effect, protect the set of keystrokes in a user interface.

Examples of some conventional keystrokes for use with menus were mentioned earlier. These include the use of <Page Up> for going back to the previous menu, <Page Down> for going to the following menu, and <Escape> for getting out of the set of menus and back to the main part of the program. Preempting the conventional usage would be counterproductive. It would impose unnecessary communication and learning costs on screen designs and users.

Other keystrokes are less clearly es-

tablished as conventional. These include use of </> to invoke a menu or command set associated with a program, as in the well-known Lotus 1-2-3 spreadsheet program. However, their use for a particular purpose may be an emerging convention.

Invocation of a command set means, as in 1-2-3, switching from one mode of operation to another. In one mode keystrokes might enter data in cells of the spreadsheet or direct the program to recalculate values in cells. In another mode the user might leave the spreadsheet and use keystrokes to call up another file or to show a pie chart that represents the data in the spreadsheet. In a database program such as dBase, the command mode is invoked for analogous purposes.

Lotus may be taking the position in two lawsuits[11, 12] that any use of </> for invocation of the command set associated with a competitive spreadsheet program is infringement of the copyright in 1-2-3. Perhaps the position is tempered by being asserted only in the context of an alleged imitation of a great many other keystroke and menu features of 1-2-3.

The seriousness of preempting an emerging convention of this type, if indeed </> is one, presumably depends on the strength of the mindset users acquire. Other keystrokes are now in use to invoke a command set. For example, the well-known Ashton-Tate dBase database management program uses <.>. Moreover, users sometimes adjust to different user interfaces when appropriately motivated to do so. Yet, where use of a particular keystroke is an established or emerging convention, a rule of law against competitive imitation would impose costs on users and designers. That rule may be unjustifiable on a cost-benefit analysis.

What has been said of keystrokes applies with equal force to other user interface techniques. Use of a beeping noise to get a user's attention, so that the user will respond to a prompt on the screen, is a common and useful device. It may be considered an audio analog of using blinking video. Another common and useful technique in user interfaces is speeding up cursor motion if an arrow key is held down for longer than a few seconds. Doubtless, vocalization conventions will develop for voice-actuated systems, and tactile conventions for touch screens. Clearly, if Lotus is willing to litigate today over the use of </>, tomorrow someone else will decide to litigate over beeps or other nonkeystroke user interface expedients.

Probably, the concern that some in the industry feel in this area is one over cumulative effect. Each item, considered alone, is of limited scope. Its preemption or removal from the public domain has a limited effect; the obstacle created is not insurmountable. The fear expressed in the industry is one of being nibbled to death, or of a thousand feather strokes somehow resulting in fatality or in the erection of barricades against innovation and competition.

The real impact of overextension of copyright protection is uncertain and difficult to quantify. However, decisions such as Softklone suggest that something more is involved here than mere figments of the imagination. It is important to strike a balance between encouragement of investment in software innovation and fettering the remainder of potential software innovators. That result is more easily prescribed than effectuated.

In the next issue I will continue to discuss copyright aspects of intellectual property protection, presenting the background of copyright law and commenting on the Copyright Office's position on screen displays. Your comments on the series so far, rejoinders or supplements concerning screen display technology matters, pertinent anecdotes, and any suggestions for future issues may be sent directly to me.

References

1. Atlantic Works *v.* Brady, 107 US 192, 199-200 (1882); E.F. Johnson Co. *v.* Uniden Corp. of Am., 623 F. Supp. 1485, 1498 n. 11 (D. Minn. 1985) (computer software innovations build on past innovations to provide new products.)
2. M. Dailey, "The 'Look and Feel' of Copyrightable Expression," *9 Eur. Intel. Prop. Rev. 234,* 234-35 (1987).
3. Digital Communications Associates, Inc. *v.* Softklone Distributing Corp., 659 F. Supp. 449, 2 U.S.P.Q.2d 1381 (N.D. Ga. 1987) (referred to hereafter as Softklone).
4. *IEEE Micro,* Vol. 6, No. 6, Dec. 1986, p. 77.
5. D.L. Hannum, "Micro Review: Three for Christmas," *IEEE Micro,* Vol. 6, No. 6, Dec. 1986, p. 84.
6. *Softklone*, 2 U.S.P.Q.2d at 1392.
7. W.O. Galitz, *Handbook of Screen Format Design,* 3rd ed., 1988, pp. 93-95.
8. R.H. Stern, "Micro Law: Software Copyright Developments," *IEEE Micro,* Vol. 7, No. 3, June 1987, pp. 81-82.
9. Discussion of a Human Factors Society report on the benefits of standards and guidelines in *SIGCHI Bull.,* Vol. 18, No. 3, Aug. 1987, p. 23.
10. Mark Ursino, *PC Week,* Aug. 25, 1987.
11. Lotus Development Corp. *v.* Paperback Software, Inc., D. Mass., Civ. No. 87-0076K.
12. Lotus Development Corp. *v.* Mosaic Software, Inc., D. Mass., Civ. No. 87-0074K.

continued from p. 96

spacer around the edges. These steel parts are welded together by a laser.

Conventional plastic cards are embossed to create the raised letters for imprinting. This method cannot be used for the Supersmart Card because the pressure would destroy the components. So Toshiba developed a method that applies strips of plastic to the stainless-steel surface, then engraves the raised characters with a high-speed, numerically controlled milling machine.

What do you put into the Supersmart Card?

In addition to the 8-bit microprocessor, it contains 16 Kbytes of ROM for the permanent program and 8 Kbytes of static RAM for data. The 5 × 7 liquid-crystal-display dot matrix is 16 symbols wide. The card's major components are implemented in very high density, low-power CMOS. Two paper-lithium batteries produce three volts for about three years of average use.

The card holds two contacts on the card, as specified by ISO standards. An asynchronous receiver/transmitter permits serial communication to a card accepting device.

How much will all this cost?

We predict the manufacturing cost will be about $20 each in very large quantities.

Do any significant technical problems remain to be solved?

No, not really. Toshiba has manufactured about 1,000 units, which a few Visa staff members in the United States and Japan have been using in place of the current magnetic-stripe cards. There have been some small reliability problems, but Toshiba expects to have them corrected by this fall.

How is the Supersmart Card programmed?

In assembler. The code is mask-programmed into the read-only memory by the manufacturer. The issuing bank can adapt the program to its requirements by entering data in files accessible only to it. The consumer can carry out a variety of operations by responding to prompts that appear in the display.

How does the Supersmart Card work?

The card is normally turned off, to conserve power. When the cardholder pushes the Yes key, the card displays time and date alternately. Now, pressing the Visa key activates the credit-card services. Then the cardholder must correctly enter a secret 4- to 12-digit Personal Identification Number. This number may have mnemonic significance as the keys also carry letters. The PIN number does not appear in the display, only asterisks to indicate that contact has been made.

By pressing the Next or Back keys, the consumer may scroll through the financial services the card offers, such as: Make a purchase? See amount available? See purchases? Add to account? Select currency?

The consumer selects one of the services by pressing the Yes key at the time that service is in the display. If the user selects Make a Purchase, the display prompts entry of the amount of the proposed transaction. The program compares the amount with the current balance. If it covers the purchase, the card displays an authorization number. The consumer shows the card to the store clerk who copies the amount and the authorization number to the sales slip.

The issuing bank periodically replenishes the consumer's credit balance. One method is to show this amount on the consumer's monthly statement, together with a cryptographic code enabling the consumer to add it to his card balance.

What do the field tests show?

Few field tests have been completed and most have not even started yet. Visa International began field tests in Japan in 1988 in cooperation with seven Japanese companies. In the US Visa International plans to use approximately 5,000 to 10,000 cards in field tests to begin near the end of 1989. So we really don't know what the results will be.

Still, we have some preliminary impressions. Some people don't want to be put in the position of keying in their PINs, purchase amounts, or other information under the pressure of the waiting sales clerk. These seem to be the same people who don't like to use automated teller machines, who reject personal computers, object to automatic direct deposit of their paychecks, and have a hard time making a VCR behave.

Then there are the people interested in innovative ideas. They place a high value on services that reduce the time and effort devoted to routine tasks. Frequent international travelers seem to fall in this class, for example.

How do merchants feel about the card?

A lot of them have reacted very favorably. They thought it was very prestigious and had a unique, differentiated image. Because it is different, they believed it would appeal to many people.

Some merchants did not like the necessity of dual procedures during the changeover period—telecommunications authorization with existing cards and self-contained authorization with the new card. Staff would have to be retrained.

Will the Supersmart Card reduce transaction time?

Where the consumer is entering a PIN and the purchase amount manually and the clerk is copying the authorization number and amount to the sales slip, a certain amount of time will be taken up. But time will be saved by not having to make an authorization call. The tests will tell us more about these times.

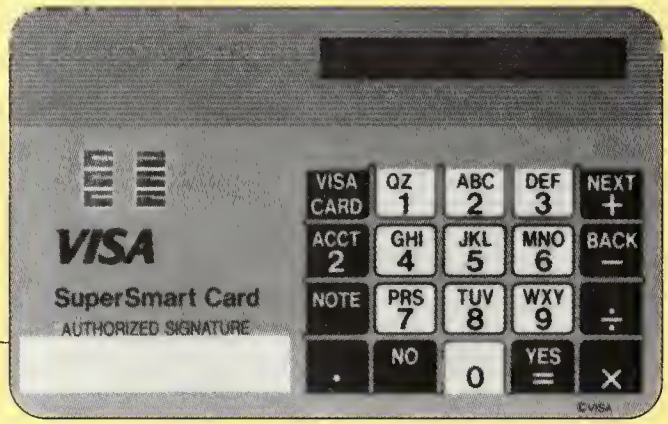
Will it reduce processing cost?

In the United States the telecommunications cost of obtaining authorization is relatively low, but in many other countries such as those in western Europe this cost is quite high. Self-contained authorization procedures could potentially reduce communications cost.

Getting the magnetic-stripe technology into widespread use took about seven years. When will we see extensive use of the Supersmart Card?

How fast Visa moves in the US will depend on the results of the field experiments. A time frame of five to seven years to test user acceptance is not unreasonable. It is possible that for a long time to come use of the Supersmart Card will be limited to niche markets—consumers who welcome innovative services or who have special needs for such services, such as international travelers.

Micro View



The size of a standard credit card, Visa's Supersmart Card stands alone. Data can be keyed in and read out from the liquid crystal display and then can be interfaced to terminals via the contacts above the word, Visa.

The road to the Supersmart Card

Ware Myers
Contributing Editor

Designers call on the microprocessor to control a wide variety of applications these days—applications that serve a very definite need. Some proposed applications, however, may move beyond a specific need and add advanced benefits that users quite possibly may reject. A case in point, and one that is now undergoing field tests to determine user interest, is advanced credit-card technology.

This technology started with the plastic card and then evolved into the magnetic-stripe card with memory. A smart card already exists—in Europe. Even a Supersmart Card exists—in Japan, as a hardware prototype. Both the smart card and the Supersmart Card contain a microprocessor, memory, and other electronic elements. The smart card must use a chip-reading terminal; the Supersmart Card with its own keyboard and display need not.

Nevertheless, most of the world and particularly the US are still in the era of the magnetic-stripe card—hundreds of millions of them are in circulation. You probably have several of them in your pocket as you read this. The technology for a more advanced card is ready. The question is—are the banks, merchants, and consumers of the world ready to embrace it? In the US, for example, Mastercard has run smart-card field tests but decided against using the technology there.

What does advanced technology have in store for consumers everywhere when they are ready to embrace it? To find the answers to these questions, *IEEE Micro* talked with Gretchen McCoy, project manager for the Supersmart Card in Visa International's Planning Division, San Mateo, California. This division has responsibility for new Visa products and services in each country. A graduate of the University of California at San Diego, McCoy has been project manager for the point-of-sale terminal standardization program, the member-controlled authorization service, and the remote PC data capture system.

First of all, tell us what you mean by the term, Supersmart Card?

At first glance the Supersmart Card looks like the familiar credit card. When you pick it up, however, it turns out to be noticeably heavier than its plastic predecessor. When you turn it over, you see a keyboard and liquid crystal display reminiscent of a pocket calculator. But it is more than a plastic credit card with a calculator.

How much more?

Well, in addition to a four-function calculator, it holds a real-time clock and calendar. Several notepad sections in its data memory allow the consumer to store telephone numbers, passport numbers, addresses, and all the other important numbers that tend to end up on scraps of paper in wallets or purses. This feature appeals to consumers we have talked with.

The Supersmart Card performs all the functions of the traditional credit card. In addition, it can keep track of the running balances in several accounts. It can keep these accounts in the cardholder's home currency or in the currency of the country he or she is traveling in at the moment. Loaded with the current currency-conversion table, it can convert from one currency to another.

The Supersmart Card could be used for making airline reservations, buying and selling stocks and bonds, and authorizing telemarketing transactions—our Japanese field test is experimenting with these functions. The airline-reservation system would automatically load the consumer's itinerary into one of the notepads.

As a prepayment card, it could be used to access copying machines, pay bridge tolls, enter movie theaters, or pay for fast food. The applications are limited only by the imagination of consumers and merchants—and the investment cost of some kind of a card reader on all those copying machines and toll booths.

How did you get all this capability into something as small as a credit card?

It wasn't easy. First, our manufacturer, Toshiba, had to make all the components thin enough to fit within the 0.76 ±0.08-mm thickness of the International Standards Organization credit-card standard. That involved designing ultrathin chips, batteries, and printed circuit board. Then, because the card must withstand the pressure of sales-slip imprinting devices, the components had to be fitted between two sheets of stainless steel, separated by a stainless-steel

continued on p. 95

0272-1732/89/0800-0096$01.00 © 1989 IEEE

IEEE COMPUTER SOCIETY

A member society of the Institute of Electrical and Electronics Engineers, Inc.

Computer Society Offices

Headquarters Office

1730 Massachusetts Ave. NW
Washington, DC 20036-1903
Phone (202) 371-0101
Telex: 7108250437 IEEE COMPSO
Fax: (202)728-9614

Publications Office

10662 Los Vaqueros Cir.
Los Alamitos, CA 90720-2578
Membership and General Information: (714) 821-8380
Publication Orders: (800) 272-6657
Fax: (714) 821-4010

European Office

13, Ave. de L'Aquilon
B-1200 Brussels, Belgium
Phone: 32 (2) 770-21-98
Fax: 32 (2) 770-85-05

Asian Office

Ooshima Building
2-19-1 Minami-Aoyama, Minato-ku
Tokyo 107, Japan
Phone: 81 (3) 408-3118
Fax: 81 (3) 408-3553

Use the Reader Service Card to obtain information on:

- Membership application—student #203, others #202
- Perodicals subscription form for individuals #200
- Periodicals subscription form for organizations #199
- Publications catalog #201
- Standards working groups list #195
- Compmail+ international electronic mail/database brochure #194
- Technical committee list/application #197
- Chapters lists, start-up procedures—student/regular #193
- Student scholarship information #192
- Awards description/nomination forms #198
- Volunteer leaders/staff directory #196
- IEEE senior member application #204

Purpose

The IEEE Computer Society advances the theory and practice of computer science and engineering, promotes the exchange of technical information among 100,000 members worldwide, and provides a wide range of services to members and nonmembers.

Membership

Members receive the acclaimed monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others seriously interested in the computer field.

Publications and Activities

Computer. An authoritative, easy-to-read magazine containing tutorial and in-depth articles on topics across the computer field, plus news, conferences, calendar, interviews, and new products.

Periodicals. The society publishes six magazines and four research transactions. Refer to membership application or request information as noted above.

Conference Proceedings, Tutorial Texts, Standard Documents. The Computer Society Press publishes more than 100 titles every year.

Standards Working Groups. Over 100 of these groups produce IEEE standards used throughout the industrial world.

Technical Committees. More than 30 TCs publish newsletters, provide interaction with peers in specialty areas, and directly influence standards, conferences, and education.

Conferences/Education. The society holds about 100 conferences each year and sponsors many educational activities, including computing science accreditation.

Chapters. Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.

European Office

Payments for Computer Society membership and publication orders are accepted by checks in Belgian, British, German, Swiss, or US currency. Checks in US funds must be drawn on a US bank. Payment may also be made by American Express, Eurocard, MasterCard, or Visa credit cards.

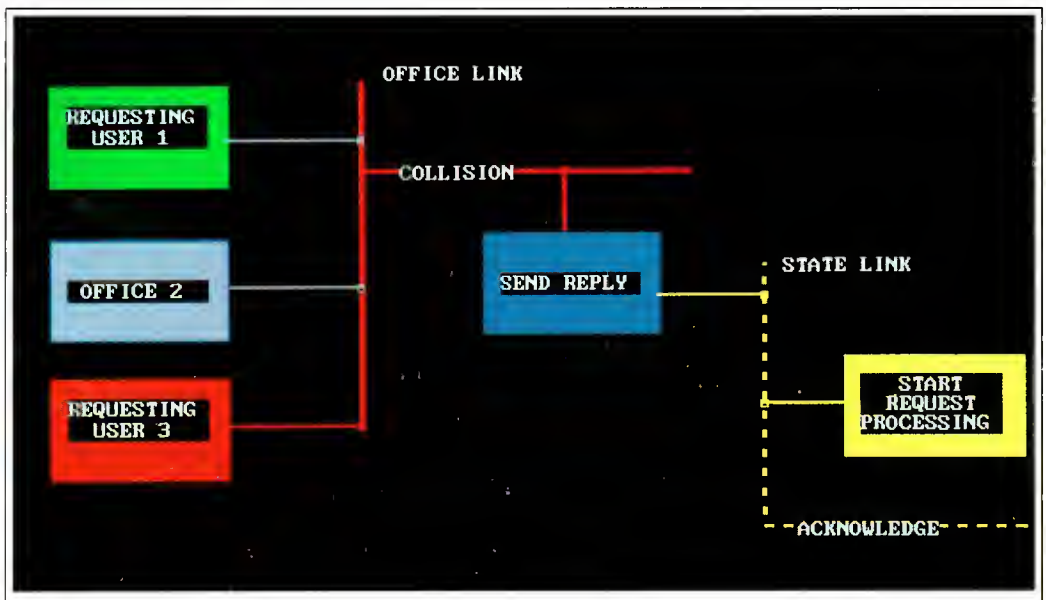
Asian Office

Payments for Computer Society membership and publication orders are accepted by checks in Japanese or US currency. Checks in US funds must be drawn on a US bank. Payment may also be made by electronic fund transfer to the Bank of Tokyo, Akasaka Branch, Toza acct. 0767956; the credit receiver is the IEEE Computer Society Headquarters Office. Payment may also be made by American Express, Eurocard, MasterCard, or Visa credit cards.

Ombudsman

Members experiencing problems — magazine delivery, membership status, or unresolved complaints — may write to the ombudsman at the Publications Office.

NEW FOR NETWORK ANALYSTS



Realistic simulation of your network--quick results, no programming

NETWORK II.5 now predicts performance of Ethernet, Token Ring, Token Bus, or FDDI

Free trial and, if you act now, free training

NETWORK II.5 uses simulation to predict your network performance. You simply describe your network and workload.

Animated simulation follows immediately--no programming delays.

Easy-to-understand results

You get an animated picture of your network. System bottlenecks and changing levels of utilization are apparent.

You can simulate some portions of the network at a detailed level and others at a coarser level.

Your reports show response times, messages delivered, messages lost, device utilization, and queueing statistics.

Computers with NETWORK II.5

NETWORK II.5 is available for most PC's, Workstations, and Mainframes.

Your network simulated

You can analyze any LAN or other computer-communication network. Industry standard protocols such as FDDI and IEEE Standard 802.X are built-in. Others can be modeled.

You can easily study the effect of changing network parameters or even network protocols.

Seeing your network animated increases everyone's understanding of its operation and builds confidence in your results.

Free trial information

The free trial contains everything you need to try NETWORK II.5® on your computer. For a limited time we also include free training **--no cost, no obligation.**

Call Paul Gorman at (619) 457-9681, FAX (619) 457-1184. In Europe, call Richard Eve on (01) 528-7980, FAX (01) 528-7988.

Free trial offer

See for yourself how NETWORK II.5 quickly answers network performance questions.

Limited offer--Act now for free training.

☐ Send information on your Special University offer.

Name

Organization

Address

City State Zip

Telephone Computer

IEEE MICRO

Return to:
CACI Products Company
3344 North Torrey Pines Court
La Jolla, California 92037
Call Paul Gorman at (619) 457-9681.
FAX (619) 457-1184.

In Europe:
CACI Products Division
Regent House, 89 Kingsway
London WC2B 6RH, United Kingdom
Call Richard Eve on (01) 528-7980.
FAX (01) 528-7988.